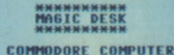
COMMODORE 64

Rita Bonelli

il basic

GRUPPO
EDITORIALE
JACKSON

COMMODORE 64

il basic

di

Rita Bonelli

GRUPPO
EDITORIALE
JACKSON
Via Rosellini, 12
20124 Milano

INDICE

PREFAZIONE

Il COMMODORE 64 sta avendo larga diffusione sul mercato e molte persone chiedono manuali che li aiutino a diventare degli esperti per poter sfruttare in pieno le possibilità del loro calcolatore.

Dopo essermi occupata del VIC 20 mi fa piacere dedicarmi a questo suo "fratello maggiore". Così è nata l'idea di preparare una serie di manuali dedicata al COMMODORE 64. Ne ho parlato con alcune persone che collaborano volentieri con me, con l'Editore, e abbiamo cominciato a lavorare. Questo è il primo volume della serie; seguiranno a breve termine, uno dedicato al trattamento dei file su disco e cassetta e uno dedicato alla grafica e al suono. Poi vedremo. Su ogni volume vengono indicati i nomi degli autori; cercherò di essere sempre presente, almeno in parte.
La filosofia che sta alla base di questi libri è sempre la stessa, si tratta di usare il calcolatore per "imparare a usarlo e quindi a programmarlo". Come sempre le cose dette vengono accompagnate da molti esempi e viene indicato un metodo per capire cosa fa la macchina, sia dal punto di vista hardware che software.

Lo scopo di questo volume, che si può definire "un libro di programmi per imparare a programmare", è quello di insegnare a scrivere programmi in BASIC, imparando a gestire il video e la stampante e facendo un buon uso della memoria del calcolatore. Tratta inoltre della memorizzazione dei programmi su disco e su cassetta. Sono lasciati volutamente da parte, in quanto argomento dei due prossimi volumi già annunciati, i problemi inerenti il trattamento dei file di dati su disco e su cassetta, la grafica e il suono.
Rimangono molti altri argomenti che sarebbe interessante approfondire, ma non si può fare tutto in poco tempo.

Il libro può essere letto in diversi modi; gli esperti sanno già cosa andare a cercare e quindi non hanno bisogno di consigli. I principianti troveranno un pò difficili alcuni approfondimenti ma, dato che tutto quello che dico è accompagnato da esempi, potranno trovare un valido aiuto. Alcuni argomenti, se risultano ostici ad una prima lettura, potranno essere riesaminati più tardi.

Nel Capitolo 1 si ha una panoramica dei diversi argomenti. Il Capitolo 2 è dedicato al linguaggio; il suo indice riporta in ordine alfabetico le parole chiave del Basic, consentendo una rapida ricerca. Nel Capitolo 3 si approfondisce l'uso della tastiera e del video. Il Capitolo 4 fornisce le informazioni necessarie per usare disco e

cassetta per memorizzare programmi, inoltre insegna l'overlay dei programmi e fornisce utili suggerimenti per migliorare il proprio lavoro sul calcolatore. Il Capitolo 5 è dedicato alla stampante e insegna a eseguire la copia del video in diversi modi. Nel Capitolo 6 ci occupiamo della costruzione del programma. Nel Capitolo 7 vengono passati in rassegna i codici e i numeri del calcolatore. Il Capitolo 8 è dedicato alla memoria, vengono indicate le possibili configurazioni del COMMODORE 64 e sono riportate le tabelle di assegnazione degli indirizzi. Nel Capitolo 9 ci occupiamo degli errori. Completano il libro, l'Appendice A dedicata alla tastiera e l'Appendice B che tratta l'argomento del Basic compilato.

Termino esprimendo la speranza che il mio lavora serva a soddisfare le aspettative dei lettori che appartengono alla categoria dei "curiosi", cioè di coloro che non vogliono solo giocare con il calcolatore, ma lo vogliono anche conoscere.

(Rita Bonelli)

L'autrice ringrazia James Bachmann, Amministratore Delegato, e Sergio Messa, Direttore Generale della Commodore Italiana s.p.a., per avere messo a disposizione le apparecchiature e la documentazione necessarie alla realizzazione dell'opera.

Tutti i programmi che compaiono nel testo sono stati provati sul calcolatore e sono stati memorizzati su un dischetto. L'editore fornisce a richiesta copie di questo dischetto.
I numeri esadecimali che compaiono nel testo, salvo avviso contrario, sono contraddistinti da una H finale.

CAPITOLO 1

UNO SGUARDO D'INSIEME

1.1 IL CALCOLATORE

Dopo aver effettuato le operazioni di montaggio suggerite dal manuale contenuto nella scatola ti trovi davanti la tastiera e il video; dai corrente e vedi comparire sul video:

```
    **** COMMODORE 64 BASIC V2 ****

 64K RAM SYSTEM 38911 BASIC BYTES FREE

READY.
```

con un quadratino lampeggiante sotto la prima lettera della parola READY. Tale quadratino lampeggiante si chiama CURSORE.
Cosa è successo? Il COMMODORE 64 ti avvisa che è pronto per lavorare e attende i tuoi comandi. Vediamo cosa sta sotto questo.

Il calcolatore è una macchina costruita per funzionare in un modo ben definito, una macchina elettronica. Il modo nel quale essa è stata costruita la rende molto versatile, al punto che non sembra solo una macchina.

La parte fisica del calcolatore è il suo HARDWARE. Esso si schematizza nei seguenti componenti:

.. Unità Centrale
.. Unità di Entrata
.. Unità di Uscita.

UNITA' ENTRATA → UNITA' CENTRALE → UNITA' USCITA

Figura 1.1 Schema del calcolatore

Nell'Unità Centrale (CPU-Central Processing Unit) si distinguono le seguenti parti:

.. Unità di Controllo
.. Unità Aritmetico Logica
.. Memoria

MEMORIA	UNITA' DI CONTROLLO	UNITA' ARITMETICO LOGICA

Figura 1.2 Schema dell'unità centrale del calcolatore

L'Unità Centrale è il calcolatore; esso ha bisogno almeno di una Unità di Entrata e di una Unità di Uscita per comunicare con l'uomo.

Il calcolatore sa fare una sola cosa: eseguire le istruzioni che vengono immagazzinate nella sua memoria; queste istruzioni costituiscono il suo PROGRAMMA di lavoro.

L'UNITA' DI CONTROLLO fa funzionare il calcolatore eseguendo il programma che sta nella MEMORIA e attivando l'UNITA' ARITMETICO LOGICA, quando serve.

I programmi costituiscono il SOFTWARE del calcolatore, quello che lo rende versatile. Infatti i PROGRAMMI possono essere immagazzinati nella memoria uno per volta, scegliendo ogni volta quello che serve.

L'UNITA' DI ENTRATA serve per comunicare con il calcolatore: per far entrare nella memoria i programmi e i dati necessari per lavorare. L'UNITA' DI USCITA serve per ricevere dal calcolatore i risultati dei lavori eseguiti.

Nel tuo COMMODORE 64 l'Unità centrale sta sotto la tastiera che vedi, l'Unità di entrata è la tastiera e l'Unità di uscita è il video che colleghi alla tastiera. Se vuoi puoi collegare al calcolatore anche altre unità di entrata e uscita; ne parleremo più avanti.

La MEMORIA è fondamentale e ce ne occupiamo quel tanto che basta per poter proseguire. La scritta che compare all'accensione del calcolatore ti dice che hai "38911 BASIC BYTES FREE" (38911 BASIC BYTES LIBERI). La parola BYTES è oramai entrata nel gergo informatico e non viene di solito tradotta. Il byte è l'unità elementare di informazione che può stare nella memoria del calcolatore; esso viene usato come unità di misura della memoria.
La memoria del calcolatore è formata da tante caselle (cellette, contenitori); come un grande foglio di carta disegnata a rettangoli. Ogni casella puoi immaginarla come una scatolina rettangolare, ulteriormente suddivisa in 8 piccoli scomparti.

BYTE

BIT: 7 6 5 4 3 2 1 0

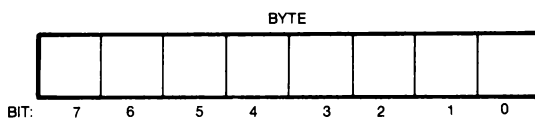



Figura 1.3 Schema di una cella di memoria, con il byte formato da 8 bit

Ognuno degli 8 scomparti può contenere un solo segno, anzi una cifra, che può essere o zero o uno. Il contenuto di ogni scomparto si chiama BIT. Un BYTE è formato da 8 BIT. Il BIT è l'unità minima di informazione per il calcolatore, però non possiamo somministrargli un singolo bit, siamo obbligati a fornirgliene 8 tutti insieme, un intero byte. Se immaginiamo la memoria del calcolatore come un grande foglio di carta, lo vediamo pieno di zeri e di uno.
All'inizio il calcolatore ti dice che hai a disposizione 38911 cellette; puoi quindi sbizzarrirti a scrivere 38911x8 cifre 0 o 1. Devi imparare a riempire le cellette in modo che quello che ci scrivi abbia un significato per il calcolatore.

Dal momento che tu sei abituato a scrivere numeri, parole, simboli per risolvere dei problemi, deve esistere un modo per comunicarli al calcolatore usando solo degli zeri e degli uno. Si tratta ovviamente di usare un codice, nel quale ogni ottetto di zeri e uno abbia un diverso significato.
Combinando in tutti i modi possibili 2 simboli (0 e 1) in gruppi di 8, si ottengono 256 disposizioni diverse. In conseguenza in un byte si possono rappresentare 256 cose diverse; le chiamiamo caratteri. Nel nostro caso si tratta del CODICE ASCII; esistono anche altri tipi di codici.
Nel Capitolo 7, Paragrafo 4, sono riportati i codici ASCII; potrai vedere che:

00110000 corrisponde alla cifra 0
01000001 corrisponde alla lettera A.

In realtà nella tabella ivi riportata, prodotta da un programma, compaiono solo i codici stampabili (cioè quelli che corrispondono a un vero carattere). Nella tabella il codice ASCII è riportato come numero decimale, esadecimale e binario, vicino al carattere corrispondente. Inoltre, per ogni carattere stampabile, è riportato il codice usato per farlo apparire sul video; codice che risulta diverso dal codice ASCII e viene chiamato D/CODE (display code). Vediamo associato alla cifra zero il numero decimale 48 e alla lettera A il numero decimale 65. Abitualmente si fa riferimento ai codici decimali ASCII, la rappresentazione binaria risulta fastidiosa e quella esadecimale non troppo agevole.

Per il calcolatore si è dovuto usare un'aritmetica diversa da quella decimale, la binaria. In essa valgono le stesse regole dell'aritmetica decimale riguardo al valore posizionale delle cifre in un numero, solo che le cifre disponibili sono due invece che dieci, e il valore posizionale delle cifre si calcola in base alle potenze di due invece che alle potenze di dieci.
Nel nostro BYTE i pesi da attribuire alle singole cifre (BIT) sono:

BYTE

BIT:	7	6	5	4	3	2	1	0
PESI:	128	64	32	16	8	4	2	1

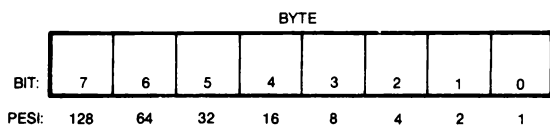



Figura 1.4 Valori posizionali dei bit in un byte

che corrispondono, partendo dal bit più a destra, alle seguenti potenze di 2:

$2^0 = 1$ $2^1 = 2$ $2^2 = 4$ $2^3 = 8$

$2^4 = 16$ $2^5 = 32$ $2^6 = 64$ $2^7 = 128$

Ne consegue che il numero più piccolo che si può scrivere in un byte è zero, mentre il più grande è 255; infatti:

$$0 \times 2^7 + 0 \times 2^6 + 0 \times 2^5 + 0 \times 2^4 + 0 \times 2^3 + 0 \times 2^2 + 0 \times 2^1 + 0 \times 2^0 = 0$$

$$1 \times 2^7 + 1 \times 2^6 + 1 \times 2^5 + 1 \times 2^4 + 1 \times 2^3 + 1 \times 2^2 + 1 \times 2^1 + 1 \times 2^0 =$$

$$128 + 64 + 32 + 16 + 8 + 4 + 2 + 1 = 255$$

Possiamo, per il momento, concludere che in un byte si può rappresentare un numero che va da 0 a 255, e che ognuno di questi 256 numeri può essere interpretato come un carattere (vedi Capitolo 7).

Esiste il problema di distinguere una parte della memoria del calcolatore da un'altra, cioè di individuare i singoli bytes che la compongono.
Questo problema è stato risolto nel modo più semplice possibile; è stato assegnato ad ogni byte un indirizzo numerico. L'assegnazione degli indirizzi è stata fatta partendo da zero, secondo la successione dei numeri naturali.
Nel nome del tuo calcolatore è riportato il numero 64 perchè esso ha una memoria formata da 64K bytes. Dove K è una costante usata come abbreviazione per misurare la memoria del calcolatore; essa corrisponde al numero 1024. In conseguenza il tuo calcolatore ha una memoria formata da 64x1024 byte, cioè da 65536 bytes. Gli indirizzi con i quali vengono individuati singolarmente i bytes vanno da 0 (zero) a 65535. In informatica si comincia sempre a contare da zero, questo serve a risparmiare un bit. La quantità di memoria gestibile da un calcolatore dipende dalle caratteristiche del microprocessore che costituisce la sua struttura portante, in particolare dalla capacità di indirizzamento che esso ha. Nel nostro caso il microprocessore può gestire indirizzi il cui valore massimo è 65535. Per formare gli indirizzi vengono usati due bytes; il primo fornisce l'indirizzo della PAGINA di memoria, il secondo l'indirizzo del BYTE nella pagina. Sono disponibili 256 pagine di 256 bytes ciascuna (256x256=65536).

A questo punto, tornando alla scritta iniziale, viene subito la domanda: dove sono andati a finire parte dei 65536 bytes di memoria, dato che ne restano liberi solo 38911?

Prima abbiamo definito il calcolatore come una macchina che sa solo eseguire le istruzioni che sono immagazzinate nella sua memoria. Vediamo dunque in cosa consiste il funzionamento automatico del calcolatore. Per poter esporre brevemente questo argomento dobbiamo chiarire alcuni termini. Si chiama:

REGISTRO, una zona di memoria, formata da uno o da due bytes consecutivi, nella quale sta memorizzato un numero. Se il registro consiste in un solo byte, esso può contenere un numero che va da 0 a 255. Se esso consiste in due bytes, il valore numerico contenuto si ottiene sommando il contenuto del primo byte al contenuto del secondo moltiplicato 256. In questo secondo caso il numero massimo contenuto nel registro può arrivare a 255 + 255x256 = 65535.

CONTATORE DEL PROGRAMMA (PC - Program Counter), un registro che contiene sempre l'indirizzo del primo byte dell'istruzione di programma che deve essere eseguita.

REGISTRO DELL'ISTRUZIONE, un registro nel quale viene trasferita, prelevandola dalla memoria, l'istruzione che deve essere eseguita.

ACCUMULATORE, un registro che viene usato per effettuare le operazioni aritmetiche.

FLAG, un registro o una parte di registro (anche solo un BIT) che viene usato per ricordare il tipo di risultato di una operazione svolta.

Questi registri, insieme ad altri, stanno nella Unità Centrale del Calcolatore. L'automatismo del calcolatore consiste in questo:

1) viene prelevata dalla memoria l'istruzione il cui indirizzo sta nel Contatore del Programma e trasferita nel Registro dell'Istruzione;

2) il Contatore del Programma viene incrementato in modo da contenere l'indirizzo della successiva istruzione (le istruzioni sono immagazzinate in memoria una di seguito all'altra per indirizzi crescenti);

3) viene eseguita l'istruzione, con tutte le implicazioni che essa comporta, tipo prelevamento di dati dalla memoria e simili;

4) si torna al punto 1).

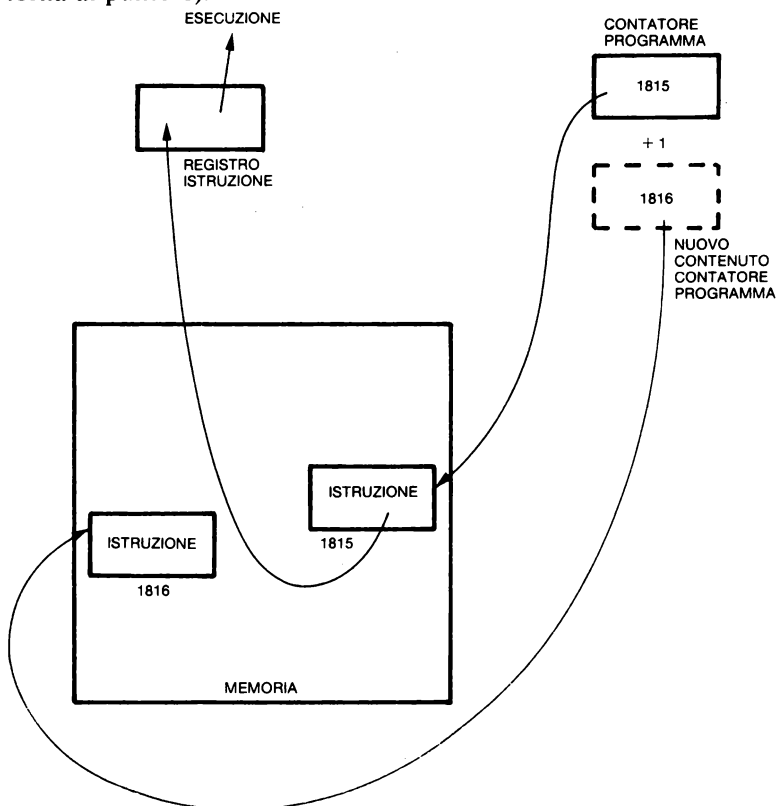



Figura 1.5 Schema del funzionamento automatico del calcolatore

Da quanto detto sembrerebbe che un programma per il calcolatore possa solo procedere per indirizzi crescenti; questo non è vero, dal momento che esistono delle istruzioni capaci di modificare il contenuto del Registro Contatore del Programma.

Per far partire il calcolatore si deve quindi scrivere nel Contatore del Programma un numero, l'indirizzo della prima istruzione da eseguire, dopo aver naturalmente memorizzato il programma, e dare lo START. Per raggiungere questo scopo ci deve essere qualche bottone, qualche tasto, che ci consenta di accedere al Contatore del Programma e alla memoria e un dispositivo per la partenza.
Guardando il COMMODORE 64 tu non vedi queste cose; quando dai corrente appare la scritta iniziale. Questo significa che il tasto di accensione avvia lo svolgimento automatico delle operazioni prima descritte, ponendo nel PC l'indirizzo fisso di un programma che sta già in memoria e viene mandato in esecuzione. Il tuo calcolatore è già in parte programmato e tu comunichi con esso con l'aiuto di un programma. In realtà si tratta di parecchi programmi che possono essere raggruppati con il nome di SISTEMA OPERATIVO (OS - Operating System) e

8K ROM KERNAL	57344/65535 (E000H - FFFFH)
4K RAM I/O	53248/57343 (D000K - DFFFH)
4K RAM (BUFFER)	49152/53247 (C000H - CFFFH)
8K ROM BASIC	40960/49151 (A000H - BFFFH)
8K RAM	32768/40959 (8000H - 9FFFH)
16K RAM	16384/32767 (4000H - 7FFFH)
16K RAM	0/16383 (0000H - 3FFFH)

Figura 1.6 Schema dell'utilizzo della memoria al momento dell'accensione.

suddivisi in tre gruppi:

.. GESTIONE VIDEO (SCREEN EDITOR)
... GESTIONE SISTEMA (ROUTINE KERNAL)
.. INTERPRETE BASIC (BASIC INTERPRETER).

In generale si parla solo di Sistema Operativo facendo riferimento ai primi due gruppi, e di Interprete Basic. Nel seguito chiameremo EDITOR il programma di gestione del colloquio video/tastiera.
Tutti questi programmi sono registrati in modo permanente in una parte della memoria del tuo calcolatore; essa prende il nome di Memoria ROM (Read Only Memory - Memoria a sola lettura). Ecco perchè la scritta iniziale dice che sono disponibili solo 38911 bytes e compare la parola BASIC a ricordare che il tuo calcolatore è pronto a ricevere da te comandi accettabili dall'Interprete Basic.

Dalla figura 1.6 puoi vedere l'utilizzo della memoria al momento dell'accensione del calcolatore. Nel Capitolo 8 si tratta diffusamente questo argomento; infatti il COMMODORE 64 può utilizzare la memoria in modo diverso da quello qui presentato. Per il momento facciamo riferimento allo schema in figura e precisiamo che nel primo blocco di RAM, quello corrispondente agli indirizzi più bassi sono utilizzati i primi 2K di memoria per i seguenti scopi:

da 0 a 1023 (0000H-03FFH) memoria di lavoro per Sistema Operativo e Interprete Basic;

da 1024 a 2023 (0400H-07E7) 1000 byte di MAPPA VIDEO;

da 2024 a 2047 (07E8H-07FFH) 24 byte per la grafica (sprite).

Per il programma BASIC dell'utente sono disponibili i 38911 byte che vanno da 2048 a 40959 (0800H-9FFFH).
Da 40960 a 49151 (A000H-BFFFH) si trova la ROM dedicata all'Interprete BASIC.
Da 49152 a 53247 (C000H-CFFFH) sono situati 4K di RAM, utilizzati per i BUFFER.
Da 53248 a 57343 (D000H-DFFFH) è mappato l'I/O.
Da 57344 a 65535 (E000H-FFFFH) sono presenti gli 8K di ROM dedicati alle routine del Sistema Operativo.

Perchè si costruiscono così i Personal Computer? Per renderne semplice l'uso da parte della gente. Facciamo un paragone: per accendere la luce in casa si usa un interruttore, non ci piacerebbe ogni volta dover stabilire i contatti tra i fili elettrici, sarebbe noioso e anche pericoloso. Analogamente sarebbe noioso dover programmare un Personal senza gli intermediari Sistema Operativo e Interprete Basic;

sarebbe anche pericoloso per le case costruttrici, che ne venderebbero pochi, e per noi che non li useremmo, mentre essi sono degli utilissimi strumenti.

Consideriamo con attenzione la figura 1.6; in essa sono schematizzate le diverse parti della memoria e il loro utilizzo. La sigla ROM è già stata spiegata; la sigla RAM significa: Random Access Memory - Memoria ad accesso random, con il significato che in questa parte della memoria si può sia leggere che scrivere (cancellando quello che vi era scritto prima). In realtà si può accedere in modo casuale a tutti gli indirizzi della memoria del calcolatore, sia ROM che RAM. Per questa ragione la sigla più corretta per quest'ultima parte della memoria è RWM, per Read Write Memory (memoria per lettura e scrittura).
Il Sistema Operativo e l'Interprete Basic non possono essere cancellati dall'utente, essi sono formati da diversi programmi che vanno in esecuzione con il procedimento automatico esposto precedentemente. Tutti questi programmi necessitano di una parte di memoria RAM, cioè di memoria nella quale si può anche scrivere, per poter lavorare, questa parte di memoria si trova da 0 a 1023.

I 38911 bytes che vengono inizialmente annunciati come liberi per il Basic sono quanto resta di memoria RAM all'utente per poter lavorare. Tu imparerai a scrivere programmi nel linguaggio BASIC, li caricherai nella memoria RAM libera e l'Interprete BASIC, sempre presente in ROM li manderà in esecuzione, servendosi di quelle parti del Sistema Operativo che via via risultino necessarie.

Prima di parlare del BASIC è bene rendersi conto, almeno in prima approssimazione, della natura di quelle istruzioni di programma di cui si è parlato a proposito dell'automatismo del calcolatore. Ogni microprocessore (e quindi ogni calcolatore) ha un gruppo (SET) di istruzioni che lo fanno funzionare e che costituiscono il suo linguaggio, quello che viene chiamato LINGUAGGIO MACCHINA. Esse sono dei codici binari, cioè dei gruppi di bit, ognuno con il suo significato. Una istruzione è formata da un codice, detto CODICE OPERATIVO, che la definisce, seguito da nessuno, uno o più operandi a seconda dei casi. In generale le istruzioni occupano uno o più byte e la lunghezza dell'istruzione o è implicitamente definita dal codice o viene riportata nel corpo dell'istruzione stessa. Durante il funzionamento automatico del calcolatore il registro Contatore del Programma si incrementa di uno se la istruzione occupa un byte, di N se la istruzione occupa N bytes.

E' possibile suddividere il SET di istruzioni in categorie; esse sono:

.. Istruzioni di Trasferimento, per trasferire contenuti di memoria da un posto all'altro;
.. Istruzioni di Calcolo, per eseguire calcoli;
.. Istruzioni di Confronto, per confrontare tra loro contenuti di memoria;
.. Istruzioni di Scelta, per far procedere il programma da un indirizzo o da un altro a seconda dello stato di un FLAG;

.. Istruzioni di Salto, per far proseguire il programma da un nuovo indirizzo, invece che in sequenza;
.. Istruzioni di Lettura, per ricevere dati o programmi dall'esterno;
.. Istruzioni di Scrittura, per mandare informazioni all'esterno.

In una istruzione di trasferimento gli operandi sono l'indirizzo di memoria da cui prelevare e quello in cui memorizzare. In una istruzione di calcolo, per esempio di somma, gli operandi sono i due addendi. In una operazione di salto l'operando è l'indirizzo dal quale deve proseguire il programma.

Le istruzioni lavorano sui registri della Unità Centrale e sulla memoria; i riferimenti si ottengono mediante gli indirizzi.

Dopo che una istruzione è stata prelevata dalla memoria e portata nella CPU nel Registro della Istruzione, essa viene controllata. Se il codice operativo non viene riconosciuto, cioè non fa parte del linguaggio macchina del microprocessore, il calcolatore si ferma e segnala un errore; infatti esso non sa cosa deve fare.

Il Sistema Operativo e l'Interprete Basic sono scritti in linguaggio macchina, ma tu non hai bisogno di imparare il linguaggio macchina per usare con profitto il tuo calcolatore. Basta imparare le regole d'uso del Sistema Operativo e dell'Interprete Basic.

Torniamo al CURSORE LAMPEGGIANTE che appare sotto la, ormai ben nota, scritta iniziale.

Nel tuo calcolatore sta funzionando il programma EDITOR; esso è in attesa di LEGGERE i tasti che tu premi sulla tastiera. Prova a scrivere:

CIAO COMMODORE 64

e a premere il tasto con sopra scritto RETURN. Vedi apparire:

?SYNTAX ERROR
READY.

Infatti il programma che sta funzionando, quando tu premi il tasto RETURN, comincia ad analizzare la frase che tu hai scritto e confronta i suoi primi caratteri con le parole contenute in una tabella nella memoria, la tabella dei comandi validi; in questa tabella non è presente alcun comando che inizi con "CIAO". Il programma allora ti segnala con "?SYNTAX ERROR" che hai commesso un errore (errore di sintassi) e con "READY." che si è messo di nuovo in paziente attesa di un tuo comando.

In realtà sono successe delle cose abbastanza complicate. Quando tu premi un tasto viene ricevuto un segnale; esso viene interpretato dal programma, riconosciuto come un certo carattere e stampato sul video nella posizione dove prima stava il CURSORE, con conseguente spostamento di questo. Solo quando tu premi il tasto RETURN il messaggio scritto e rivisto sul video viene preso in considerazione dal programma. Inoltre, tu mentre scrivi puoi sbagliare e ti è possibile correggere; questo significa che ogni carattere prima di essere stampato sul video viene analizzato e alcuni caratteri particolari producono uno specifico effetto.
I caratteri battuti sulla tastiera vanno a finire in una zona dedicata di memoria, chiamata BUFFER DI TASTIERA, che ne può contenere fino a dieci. Il buffer è gestito da un registro PUNTATORE, che segnala la posizione libera. Al momento dell'accensione del calcolatore il puntatore individua la prima posizione del buffer. Il calcolatore si può trovare nello stato di attesa di un tuo carattere battuto, o può essere impegnato in altro lavoro; in quest'ultimo caso il carattere va nel buffer e viene utilizzato appena il calcolatore ha terminato il precedente lavoro.

1.2 IL BASIC

Il BASIC è un linguaggio per scrivere programmi per il calcolatore; la sigla sta per: Beginners All-purpose Symbolic Instruction Code (Linguaggio di programmazione generale per principianti).
Un programma scritto in linguaggio BASIC viene caricato nella memoria del calcolatore e interpretato da un programma già precedentemente presente nella stessa memoria, l'Interprete BASIC. Per l'Interprete Basic il programma scritto in linguaggio Basic costituisce un insieme di dati su cui lavorare.

Il linguaggio ha un suo dizionario formato da un numero limitato di parole e simboli che l'interprete riconosce.
Le parole del dizionario, nella maggior parte dei casi una sola, a volte più di una, sono le istruzioni per il PROGRAMMA INTERPRETE BASIC.
I dati su cui operare, numeri o parole, sono forniti rispettando alcune regole semplici.

Abbiamo visto nel paragrafo precedente che il calcolatore ci segnala con il cursore lampeggiante che è in attesa di comandi, ma che non tutte le parole gli vanno bene.
Se proviamo a scrivere: LIST e poi premiamo RETURN. Vediamo scendere il cursore e fermarsi dopo aver scritto: READY.
LIST è un comando; esso viene accettato alla pressione del tasto RETURN ed eseguito. L'effetto del comando è di produrre sul video la lista delle istruzioni del programma BASIC presente in memoria; se in memoria non ci sono istruzioni, esse non compaiono e vediamo solo la parola READY che segnala di aver eseguito l'ordine.

Nel seguito useremo indifferentemente i termini comando e istruzione.

Ci sono due modi per far lavorare il nostro calcolatore:

1) eseguire istruzioni scritte sul video, senza conservarle dopo averle eseguite, cioè lavorare in MODO IMMEDIATO;

2) eseguire istruzioni dopo averle memorizzate, cioè lavorare in MODO DIFFERITO.

Per lavorare nel modo 1), scriviamo una istruzione e la mandiamo in esecuzione quando premiamo il tasto RETURN. Se vogliamo eseguire la stessa istruzione un'altra volta, dobbiamo riscriverla. Per esempio, prova a scrivere:

PRINT "ciao commodore 64" e poi RETURN

vedrai apparire sul video sotto la tua istruzione:

ciao commodore 64

e poi il solito READY.

Se vuoi rifare la stessa cosa devi riscrivere la tua istruzione. Stai lavorando in modo immediato. Quello che tu scrivi resta visibile sul video, ma non è stato conservato nella parte di memoria dedicata al programma. Se continui a scrivere, il quadro video si sposta verso l'alto e le prime cose scritte scompaiono.

Per lavorare nel modo 2) scrivi invece:

10 PRINT "ciao commodore 64" e poi RETURN

alla pressione del tasto RETURN non succede alcunchè, tu vedi sul video solo la tua linea di programma. Prova ora a scrivere:

RUN e poi RETURN

vedrai comparire:

ciao commodore 64

hai cioè eseguito il tuo programma memorizzato (programma di una sola riga); hai lavorato in modo differito, ma per fare eseguire il tuo programma hai dato il comando RUN in modo immediato. Il comando RUN fa eseguire i programmi che sono nella memoria.

Se ora vuoi eseguire tante volte il tuo programma devi solo continuare a dare il comando RUN.
Se provi ora con il comando LIST, ottieni la lista della tua riga di programma.

Da quanto precede possiamo trarre questa conclusione: il sistema calcolatore decide che stiamo scrivendo istruzioni da eseguire in modo differito se esse iniziano con un numero.

Il nostro calcolatore può dunque lavorare in due modi; esso accetta quello che scriviamo quando premiamo il tasto RETURN. In quel momento analizza l'inizio della frase scritta; se inizia con un numero la conserva nella memoria del programma, se inizia con una parola e se questa è un comando valido, lo esegue.
I due modi possono essere mescolati e non si hanno inconvenienti.

Per memorizzare un programma devi scrivere le istruzioni una linea dopo l'altra numerandole progressivamente; devi stare attento a numerarle in modo ordinato. Infatti esse vengono conservate, e poi eseguite, in ordine di numero di linea crescente, anche se tu non sei obbligato a scriverle in ordine. Se scrivi prima la linea 40, poi la linea 30 e poi la linea 10, quando dai il comando LIST esse appaiono in ordine: 10, 30, 40. Questo naturalmente è molto comodo, dal momento che scrivendo si può dimenticare qualcosa. Inoltre tieni presente che, se scrivi due numeri di linea uguali, l'ultima linea scritta va a sostituirsi a quella scritta precedentemente con lo stesso numero.

A questo punto crediamo sia utile, se non lo hai ancora fatto, che tu prenda confidenza con la tastiera e il video; infatti essi sono i principali mezzi per comunicare con il calcolatore.

Prendiamo in considerazione quei tasti che ci permettono di muoverci sul video, cioè di spostare il cursore.
In alto a destra c'è un tasto con scritto CLR/HOME; se lo premi da solo vedi che il cursore va nell'angolo in alto a sinistra del video; hai eseguito il comando HOME, e il video ha mantenuto il suo precedente contenuto. Se invece premi il tasto CLR/HOME insieme a SHIFT, il video viene pulito e il cursore va nell'angolo in alto a sinistra; hai eseguito il comando CLEAR (abbreviato da CLR) insieme al comando HOME.

Quando avvengono queste cose, sta lavorando l'EDITOR; esso gestisce le operazioni tastiera/video, analizza il tasto (o i tasti premuti contemporaneamente) e agisce in conseguenza.

Consideriamo ora i due tasti situati in basso verso destra con le scritte CRSR accompagnate da frecce bidirezionali verticali e orizzontali; essi servono a spostare il cursore sul video. Quello con le frecce verticali, usato da solo fa spostare verso il

basso, usato insieme al tasto SHIFT fa spostare verso l'alto. Quello con le frecce orizzontali, usato da solo fa spostare verso destra, usato insieme al tasto SHIFT fa spostare verso sinistra. Inoltre questi due tasti producono una azione continua se vengono mantenuti premuti; hanno cioè la funzione di ripetizione (repeat) automatica.

In alto a sinistra c'è un tasto con la scritta INST/DEL; esso ha questa funzione: se usato da solo cancella il carattere prima del cursore e fa tornare indietro il cursore. Se non ci sono caratteri scritti fa solo tornare indietro il cursore. Anche esso ha la funzione di ripetizione automatica; nel tornare indietro passa alla riga precedente, quando arriva al limite sinistro del video, fino al raggiungimento della posizione HOME. Se il tasto viene usato insieme a SHIFT, esso, se il cursore si trova in posizione opportuna, provoca lo spostamento verso destra di quello che sta scritto dopo il cursore e crea uno spazio. Il tasto INST/DEL insieme a SHIFT attiva la funzione di INSerT. Non si può inserire dopo l'ultimo carattere scritto, inoltre non si può inserire superando le lunghezze limite consentite, per esempio, per le istruzioni di programma due linee del video. Anche per la funzione di INST si ha la ripetizione automatica mantenendo premuti i due tasti.
Le due funzioni DEL e INST sono di enorme utilità quando si scrive; esse consentono di correggere.

Anche la barra di spaziatura presente sulla tastiera al centro ha la funzione di ripetizione automatica.

Vediamo qualche utile esempio di uso dei tasti appena esaminati. Supponiamo di scrivere un comando in modo immediato, come:

PRINT "sto provando i tasti di controllo"

quando premi RETURN appare sotto il comando:

una linea bianca (non scritta), la frase tra virgolette nel comando PRINT, una linea bianca e il solito READY.
Prova a portarti con il cursore, usando i tasti freccia, alla fine della frase, appena prodotta dal comando PRINT, e a cancellarla con il tasto DEL. Poi risali con il cursore in modo da posizionarti in un punto qualunque della linea dove è ancora scritto il comando PRINT e premi RETURN. Vedrai di nuovo la frase che hai appena cancellato, infatti per effetto del RETURN mentre il cursore è posizionato sulla riga dell'istruzione, questa viene eseguita di nuovo.

Ora prova a premere contemporaneamente i due tasti che si trovano in basso a sinistra, cioè quello con sopra il marchio COMMODORE e lo SHIFT, vedrai cambiare le scritte che sono sul video; esse passano all'altro set di caratteri. Se premi ancora contemporaneamente i due tasti ritorni alla situazione precedente del video.

Ora se vuoi puoi fare pulizia con CLR/HOME.

Prova ora a premere contemporaneamente i tasti con la scritta RUN/STOP (a sinistra) e RESTORE (a destra); il video viene pulito, compare READY. nella seconda riga e il cursore appare sotto. In realtà la pressione di questi due tasti ripristina le condizioni del calcolatore al momento dell'accensione, senza però cancellare un eventuale programma presente in memoria.

Puoi ora provare tutti i tasti, con o senza il tasto SHIFT. Un modo semplice per provare i tasti è scrivere in modo immediato il comando PRINT, subito dopo premere SHIFT e 2, per aprire le virgolette, e poi battere i tasti voluti. Per ottenere la stampa si devono chiudere le virgolette e premere RETURN. Attenzione però a non superare due linee video! Puoi anche provare a non chiudere le virgolette prima di usare RETURN; ottieni lo stesso la stampa senza segnalazione di errore.

Vediamo ora dove vanno a finire le istruzioni di programma da eseguire in modo differito. Nella mappa di memoria presentata in figura 1.6, abbiamo visto che la parte di memoria RAM a disposizione dell'utente va da 0 a 40959, anzi, dato che i primi 2K sono occupati dalle variabili del sistema e dalla MAPPA VIDEO, l'area disponibile va da 2048 a 40959. Le istruzioni del programma Basic che tu scrivi vengono memorizzate a partire dal byte di indirizzo 2049.
Considerando la memoria sudddivisa in pagine, le variabili del sistema stanno nelle prime 4 pagine: 0, 1, 2 e 3, la memoria dedicata al video sta nelle pagine 4, 5, 6 e 7, il programma Basic comincia in pagina 8. Nel Capitolo 8 trattiamo diffusamente dell'utilizzo della memoria, per ora ci basta dire che le istruzioni sono memorizzate carattere per carattere a partire dal byte 2049.
Esse sono memorizzate mantenendole in ordine di numero di linea crescente. Questo significa che se tu scrivi per prima l'istruzione con numero di linea uguale a 100, essa viene memorizzata a partire dal byte 2049, se poi scrivi l'istruzione con numero di linea 50, quella con numero di linea 100 viene traslata in memoria verso i byte di indirizzo più alto e lascia libero il posto a partire dal byte 2049 per l'istruzione 50. Questo procedimento viene ripetuto tutte le volte che è necessario mentre tu scrivi un programma. Diciamo che, se tu scrivi le istruzioni in ordine, il sistema lavora di meno! Lo stesso procedimento di spostare verso gli indirizzi più alti parte di un programma avviene anche tutte le volte che si va a correggere una istruzione già scritta aggiungendo qualcosa che si era dimenticato. Analogamente, quando si cancella qualcosa, parte del programma viene traslata verso gli indirizzi più bassi. In conclusione, mentre tu scrivi il programma Basic, il sistema svolge un bel pò di lavoro per tenere in ordine in memoria le tue istruzioni. Quando hai terminato di scrivere il programma una parte della memoria a partire dal byte 2049 è occupata.

Nella zona di memoria usata come area di lavoro dal sistema ci sono dei byte che

servono per riassumere la situazione della zona di memoria dedicata al programma Basic. Per il momento consideriamo i byte di indirizzo 43 e 44 che costituiscono il PUNTATORE all'inizio del programma Basic. Quando si accende il calcolatore l'indirizzo puntato da questi due byte è 2049. Più avanti vedremo come si fa a LEGGERE il contenuto dei byte di memoria. Un altro puntatore che ci interessa considerare ora è quello costituito dai byte 45 e 46. Esso è il puntatore all'inizio delle VARIABILI. Quando si accende il calcolatore l'indirizzo puntato da questi due byte è 2051, cioè l'indirizzo di inizio del programma + 2. Man mano che si introducono le istruzioni del programma l'indirizzo puntato dai byte 45 e 46 si modifica, esso è sempre superiore di 2 all'indirizzo dell'ultimo byte occupato dal programma. Abbiamo usato la parola VARIABILI, con essa si intende definire l'insieme dei dati su cui lavora il programma. Nello scrivere il programma si creano dei nomi simbolici, con regole ben definite che vedremo, che servono ad individuare e distinguere tra loro i dati. Le variabili sono i contenitori dei dati. Ad ogni variabile (contenitore) viene assegnato uno spazio di memoria; l'assegnazione avviene partendo dalla memoria libera subito dopo il programma.

Facciamo un esempio; scriviamo un programma che ci chiede due numeri, definiti simbolicamente come A e B, calcola la somma dei due numeri, chiamandola simbolicamente C, e poi la stampa sul video. Siamo costretti a introdurre la parola chiave del linguaggio che serve per LEGGERE dalla tastiera un dato, essa è INPUT; quella per scrivere l'abbiamo già incontrata, essa è PRINT. Il programma è il seguente:

```
10 INPUT A
20 INPUT B
30 C=A+B
40 PRINT C
```

esso è formato da quattro linee:

.. la linea 10 legge un numero e lo mette nella variabile A;
.. la linea 20 legge un numero e lo mette nella variabile B;
.. la linea 30 esegue la somma del contenuto di A e del contenuto di B e pone il risultato nella variabile C;
.. la linea 40 stampa il risultato della somma contenuto nella variabile C.

Dopo aver scritto il programma il puntatore all'inizio delle variabili ha un valore che dipende dalla lunghezza del programma. Se ora scrivi RUN e poi RETURN, cioè mandi in esecuzione il programma, mentre il programma viene eseguito vengono create le variabili che servono, cioè viene assegnato uno spazio in memoria prima ad A, poi a B e poi a C. Alla linea 10 viene creato lo spazio per A, alla linea 20 viene creato lo spazio per B, alla linea 30 viene ricercata A in memoria dato che serve come addendo, viene cercata B in memoria, dato che serve come addendo, e

viene crato lo spazio per C che serve per contenere il risultato della operazione di somma. Alla linea 40 viene ricercato C in memoria per mandarne il contenuto sul video con il comando PRINT. Le variabili, cioè il loro nome simbolico di riconoscimento ed il loro contenuto attuale, sono memorizzate una dopo l'altra, byte dopo byte, a partire dall'indirizzo di inizio della zona variabili. Tale indirizzo è PUNTATO dai due byte di pagina 0, 45 e 46.

Nell'esempio di programma, sopra riportato, sono scritte 4 linee, numerate 10, 20, 30 e 40. Si usa l'accorgimento di numerare le linee di programma non con numeri consecutivi per facilitare eventuali inserzioni di nuove linee. Se si procede di 10 in 10 sono possibili 9 inserzioni. Nello stesso esempio si vede che le linee di programma 10, 20 e 40 iniziano, dopo il numero di linea, con una parola chiave, un comando Basic, e dopo di questa sono citati dei nomi simbolici, usati per individuare i dati su cui operare. La struttura delle istruzioni Basic è spesso di questo tipo. La linea 30, invece, inizia con il nome di una variabile che deve ricevere il risultato di un calcolo riportato dopo il simbolo uguale. In realtà il linguaggio ha una parola chiave anche per questo tipo di operazioni, che si chiama di assegnazione (variabile = espressione), tale parola è LET, ma è facoltativa e quindi può essere omessa.

Possiamo per il momento concludere che le ISTRUZIONI BASIC sono formate da PAROLE CHIAVE e da OPERANDI. Esse devono essere precedute da un numero di linea per essere eseguite in modo differito.

Ovviamente le PAROLE CHIAVE non devono essere usate come operandi. Nel Capitolo 2 prendiamo in esame le caratteritiche del linguaggio frase per frase. Possiamo parlare indifferentemente di frasi del linguaggio o di istruzioni del linguaggio.

Riteniamo utile avere ben presenti gli argomenti trattati in questo paragrafo, prima di proseguire, questo per studiare il linguaggio Basic cercando di capire cosa concretamente succede nella memoria del calcolatore.

1.3 IL PROGRAMMA

Il programma è una sequenza ordinata di istruzioni. Un problema può essere risolto scrivendo un programma per calcolatore. Non è però il programma che risolve il problema; il problema lo risolve la persona che lo studia, trova il metodo risolutivo adatto ed è capace di tradurre quest'ultimo in un programma per calcolatore. Non è detto che tutte queste cose debbano essere fatte dalla stessa persona. E' però importante notare che il metodo risolutivo di un problema può essere diverso, a seconda che si eseguano, per esempio, dei calcoli in modo manuale, o che si programmino i calcoli per essere eseguiti da un calcolatore elettronico.

Schematizziamo le fasi necessarie per risolvere un problema con il calcolatore:

1) studio del problema;

2) ricerca di un metodo risolutivo e stesura di uno schema che consenta di avere una visione completa di quello che si deve fare;

3) traduzione dello schema in un programma per il calcolatore.

Tu sai che un programma per calcolatore può essere scritto in uno dei tanti linguaggi disponibili; ogni linguaggio ha le sue caratteristiche. La fase 1) e la fase 2) non dipendono dal particolare linguaggio che viene usato nella fase 3); però è meglio aver presente anche nelle prime due fasi su quale calcolatore e con quale linguaggio si vuole lavorare. Questo consente di procedere nelle condizioni ideali per raggiungere risultati ottimali.

La fase 1) e la fase 2) vengono abitualmente chiamate ANALISI del problema. Di solito si parte con un'idea, tipo mi piacerebbe tenere la mia contabilità con il calcolatore, oppure, se uno è appassionato del gioco del Bridge, vorrei usare il calcolatore quando organizzo un torneo tra i miei amici, o altro.

Si deve arrivare a uno schema che metta in evidenza:

1a) quali sono i dati che devono essere elaborati;

2a) quali sono i risultati che si vogliono ottenere.

A questo punto si entra nella fase 2) e si deve arrivare a decidere come fare. I risultati si possono raggiungere in diversi modi e non è facile scegliere il migliore. In questa fase deve anche essere studiato il modo nel quale presentare i dati da elaborare; questo modo può portare a modifiche nei metodi di elaborazione. Dobbiamo onestamente riconoscere che non è molto facile, soprattutto per i principianti, se il problema non è semplice, superare le prime due fasi.

Quello che ti raccomandiamo è di scrivere in qualche modo quello che pensi; è molto importante imparare a prendere appunti ordinati che consentano di rivedere i ragionamenti fatti.

Quando un problema deve essere risolto servendosi di un calcolatore si deve prevedere a priori tutto; il calcolatore infatti non ha il colpo di genio che gli consenta di uscire da un impiccio. Se lo abbiamo programmato male e lo mandiamo in un vicolo cieco, non sa uscirne!

I calcolatori Personal si presentano in modo molto amichevole agli utenti e rendo-

no la programmazione più accessibile che non i grossi calcolatori, però per programmarli bene ci vuole sempre un certo impegno. Molti pensano di poter programmare sedendosi davanti alla tastiera e improvvisando; non è detto che non riescano a produrre qualcosa, magari anche di buono a volte. Noi comunque riteniamo che sia meglio mettersi davanti alla tastiera avendo vicino dei fogli di carta che riportino, il più chiaramente possibile, degli schemi di quello che vogliamo fare.

Esistono delle metodologie collaudate per schematizzare i metodi risolutivi di un problema, prima di arrivare alla stesura del codice nel linguaggio scelto per il calcolatore. Esse si basano sulla tecnica dei diagrammi a blocchi; cioè sulla stesura di disegni formati da simboli grafici diversi collegati da segmenti orientati. Ogni simbolo grafico ha di per sè un preciso significato che facilita la lettura dello schema. Inoltre, all'interno di ogni simbolo si pongono delle frasi esplicative delle operazioni simboleggiate.

I metodi risolutivi vengono spesso chiamati ALGORITMI; si dice che i diagrammi a blocchi sono la descrizione grafica degli algoritmi.

Si possono anche descrivere verbalmente gli algoritmi. Ognuno può scegliere la strada che gli risulta più congeniale.

La visualizzazione mediante schemi di una serie di operazioni da svolgere può essere usata in molte situazioni che non hanno attinenza con l'ambiente della programmazione. Di solito se ne trae un vantaggio di chiarezza e si riescono a vedere le cose in modo sintetico.

In ambiente informatico si chiama DIAGRAMMA DI FLUSSO uno schema grafico che riguarda il movimento di informazioni intorno al calcolatore. Il più semplice diagramma di flusso è quello riportato nella figura 1.7; esso può essere valido in molti casi.

DATI
INPUT

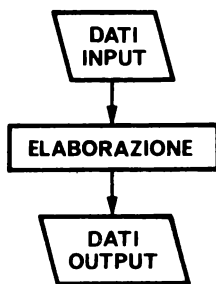
ELABORAZIONE

DATI
OUTPUT

Figura 1.7 Diagramma di flusso

Usiamo la definizione di DIAGRAMMA A BLOCCHI per lo schema grafico che rappresenta la trasformazione dei dati all'interno del calcolatore, cioè quello che fa il programma.

ELABORAZIONE

IMMISSIONE / EMISSIONE
(INPUT / OUTPUT)

CONNETTORE
(per entrata o uscita
nella stessa pagina)

CONNETTORE
(per entrata o uscita
fuori pagina)

SCHEDA PERFORATA

DOCUMENTO SU CARTA

VISUALIZZAZIONE

DISPOSITIVO
A TASTIERA

TASTIERA
DI CONSOLE

OPERAZIONI
MANUALI

OPERAZIONI
AUSILIARIE

NASTRO MAGNETICO

NASTRO PERFORATO

DISCO MAGNETICO

MEMORIA IN LINEA
(di qualunque tipo)

DECISIONE
(scelta tra strade
diverse)

TERMINALE
(inizio o fine
programma, attese)

SEGMENTI ORIENTATI
E CONNESSIONI

SOTTOPROGRAMMA

Figura 1.8 Simboli grafici per diagrammi a blocchi

Nella figura 1.8 sono riportati i simboli grafici più comunemente usati per la stesura di diagrammi a blocchi. All'interno dei simboli si scrivono delle indicazioni che personalizzano l'operazione indicata per lo specifico caso.

Ora ci proponiamo di impostare un programma che risolva il seguente problema:

CALCOLARE IL VALORE DI UN POLINOMIO IN X:

$$PL = A(n)X^{n} + A(n-1)X^{n-1} + \dots + A(1)X + A(0)$$

FACENDO VARIARE:

.. IL GRADO n;
.. GLI n+1 COEFFICIENTI A(k);
.. LA VARIABILE X.

Vogliamo arrivare a scrivere un programma in Basic per il COMMODORE 64.

Il problema è posto in modo sufficientemente chiaro. Precisiamo i dati di INPUT per il programma; essi sono:

.. il grado N del polinomio PL (numero intero);
.. gli N+1 coefficienti A(k) (numeri reali (decimali));
.. il valore della variabile X (numero reale).

Dobbiamo stabilire se vogliamo avere un programma che, fissato N e gli N + 1 coefficienti A(k), possa fornirci il valore di un determinato PL per diversi valori della variabile X, oppure se vogliamo cambiare tutto ogni volta. Ci sembra meglio produrre un programma che ci consenta di percorrere le due strade; cioè, dopo aver calcolato il valore di PL la prima volta, il programma ci chiede se vogliamo cambiare solo X o tutto. A seconda della risposta viene percorsa una o l'altra strada.

La formula (espressione) che ci mostra il polinomio non risulta comoda per il calcolo; infatti si dovrebbero calcolare N potenze della variabile X e non possiamo farlo con un comodo ciclo. Scriviamo PL in un altro modo, esponiamo cioè un algoritmo che si presta ad essere calcolato in modo iterativo (ciclico).

PL=((((XA(n)+A(n-1))X+A(n-2))X+...)X+A(1))X+A(0)

Con questo algoritmo il calcolo procede così (per maggior chiarezza usiamo

l'asterisco per indicare l'operazione di moltiplica):

```
PL=X*A(n)+A(n-1)
PL=PL*X+A(n-2)
PL=PL*X+A(n-3)
...
...
PL=PL*X+A(1)
PL=PL*X+A(0)
```

Il risultato del programma deve essere un messaggio che comunichi il valore di PL.

Passiamo ora a descrivere in modo verbale le operazioni necessarie per risolvere il problema posto.

1) Richiesta del grado N e sua introduzione;

2) Analisi del grado N; se N=0 STOP;

3) Impostazione di un ciclo per leggere gli N+1 coefficienti del polinomio e loro introduzione;

4) Richiesta del valore della variabile X e sua introduzione;

5) Calcolo ciclico del valore del polinomio;

6) Stampa del risultato;

7) Domanda circa il proseguimento del programma con lo stesso polinomio e ricezione della risposta;

8) Scelta percorso in base alla risposta; se essa è di proseguire con lo stesso polinomio si va al punto 4); se no si va al punto 1).

La precedente descrizione del programma è a GRANDI BLOCCHI; ogni punto può essere ulteriormente sviluppato arrivando alla sequenza delle singole istruzioni. E' stato necessario introdurre il punto 2) per poter uscire dal programma quando non si desidera più calcolare valori di polinomi.

Segue la descrizione grafica del programma con un diagramma a blocchi che rispetta la precedente descrizione verbale. Dal confronto potrai stabilire quale metodo preferisci.

INIZIO

RICHIESTA E INTRODUZIONE N

N = 0

SI

STOP

NO

LETTURA CICLICA N+1 COEFF. IN A(K)

RICHIESTA E INTRODUZIONE X

CALCOLO POLINOMIO

STAMPA RISULTATI

MESSAGGIO PER SCELTA-PROSE-GUIMENTO E RICEZIONE RISP.

NO

CAMBIO SOLO X

SI

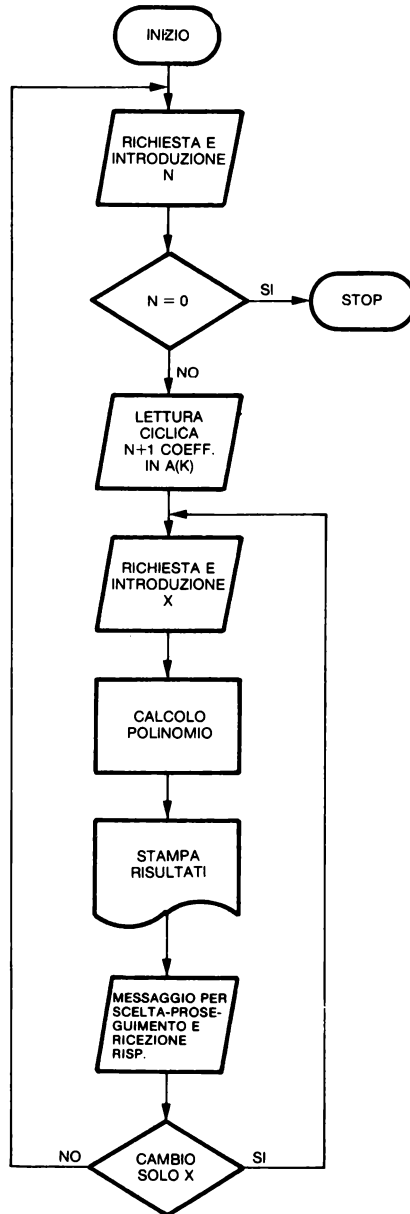



Figura 1.9 Diagramma a blocchi algoritmo calcolo valore polinomio

Dai due schemi riportati si può scendere a schemi più vicini al programma, precisando i singoli passi, ma preferiamo lasciare a te questo lavoro come utile esercizio.

Anche se per i principianti è troppo presto, facciamo seguire il listato del programma in Basic e qualche risultato su stampante. Potrai sempre tornare a questo punto più tardi.

```
1 REM CALCOLO POLINOMIO
10 OPEN4,4:PRINT#4,"     CALCOLO VALORE POLINOMIO"
20 CLOSE4:OPEN4,4
23 INPUT"GRADO POLINOMIO: N%=";N%:PRINT""
25 IFN%=0THENCLOSE4:STOP
30 DIMA(N%):GOSUB100:REM SOTTOPROGRAMMA LETTURA
35 INPUT"VALORE X: ";X
40 PRINT#4:PRINT#4,"COEFFICIENTI DEL POLINOMIO:"
45 I=0:FORK=N%TO0STEP-1
50 PRINT#4,"A(";K;")=";A(K);"   ";:I=I+1
55 IFI=4THENPRINT#4:I=0
60 NEXTK:IFI<>0THENPRINT#4
65 PRINT#4,"PER X=";X;"IL VALORE DI PL E': ";
70 GOSUB200:PRINT#4,PL:PRINT#4
75 PRINT"VUOI USARE LO STESSO POLINOMIO "
77 INPUT"RISPONDI S/N ";R$
80 IFR$="S"THEN35
85 RUN20
100 FORK=0TON%
105 PRINT"A(";K;")=";:INPUTA(K)
110 NEXTK:RETURN
200 PL=A(N%)
201 FORK=N%-1TO0STEP-1
203 PL=PL*X+A(K)
205 NEXTK:RETURN
```

RISULTATI SU STAMPANTE

```
     CALCOLO VALORE POLINOMIO

COEFFICIENTI DEL POLINOMIO:
A( 5 )= 1    A( 4 )= 8    A( 3 )=-5    A( 2 )= 3
A( 1 )=-45    A( 0 )= 90
PER X= 3 IL VALORE DI PL E':  738
```

```
COEFFICIENTI DEL POLINOMIO:
A( 5 )= 1    A( 4 )= 8    A( 3 )=-5    A( 2 )= 3
A( 1 )=-45    A( 0 )= 90
PER X= 1 IL VALORE DI PL E': 52
```

Nello sviluppare il programma in Basic abbiamo usato la tecnica dei sottoprogrammi, cioè la chiamata di piccole sequenze di programma che svolgono un determinato compito. Seguono i diagrammi a blocchi dei due sottoprogrammi usati.

SOTTOPROGRAMMA 200
PL = A (N%)
K = N% - 1
PL = PL * X+A (K)
K = K−1
K < 0
NO
SI
RETURN

SOTTOPROGRAMMA 100
K = 0
RICHIESTA E RICEZIONE A (K)
K = K+1
K ⩽ N%
SI
NO
RETURN

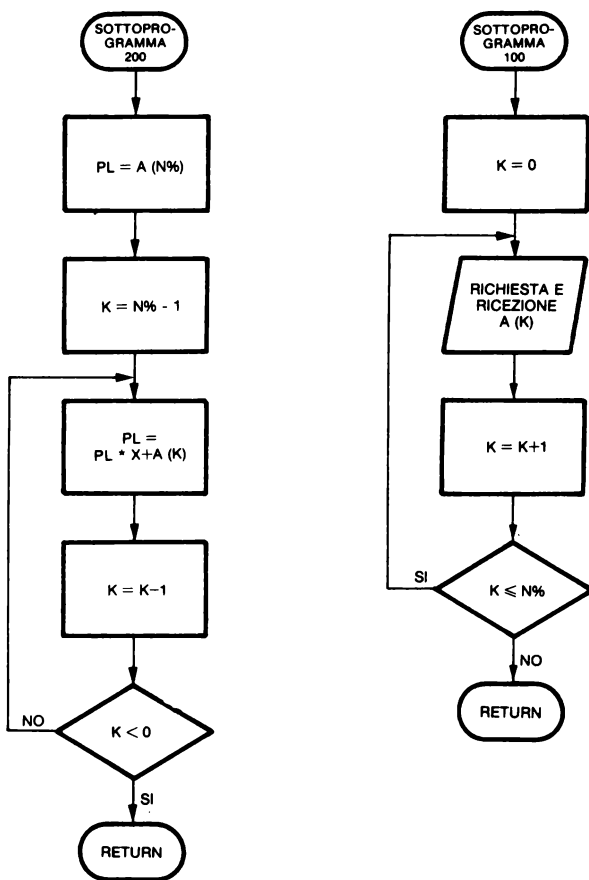



Figura 1.10 Diagrammi a blocchi dei due sottoprogrammi

Alla linea 30 abbiamo potuto usare l'istruzione DIM con N% come dimensione, cioè dimensione variabile, infatti il Basic del COMMODORE 64 lo consente. Con altro calcolatore che non consenta i dimensionamenti variabili si dovrebbe dare un dimensionamento fisso, per esempio a 100, e , dopo la lettura del grado N% del polinomio, controllare se N% < = 100.
L'uscita sulla stampante viene controllata in modo da stampare al massimo 4 coefficienti per riga; usiamo un contatore I all'interno del ciclo di stampa.
L'istruzione RUN20 alla linea 85 serve per abolire il dimensionamento, però RUN cancella anche la tabellina dei file aperti, abbiamo quindi dovuto chiudere alla linea 20 con CLOSE4 la stampante e poi riaprirla.

CAPITOLO 2

LINGUAGGIO BASIC

2.1 INTRODUZIONE

Il linguaggio Basic è di tipo:

.. interpretativo,
.. conversazionale.

La parola INTERPRETATIVO significa che un programma scritto in linguaggio Basic viene eseguito per mezzo di un altro programma che è presente in memoria contemporaneamente. Questo programma si chiama INTERPRETE BASIC e nel COMMODORE 64 si trova memorizzato in modo permanente in 8K di ROM.
La parola CONVERSAZIONALE significa che mentre tu lavori puoi interagire con il sistema calcolatore, intervenendo per modificare il programma in corso o per controllare risultati intermedi o per correggere errori che ti sono stati segnalati dal sistema. Un programma Basic in esecuzione può essere interrotto con la pressione del tasto RUN/STOP; si può intervenire con comandi eseguiti in modo immediato senza danneggiare il programma, e poi proseguire nell'esecuzione con il comando CONT.
Queste due caratteristiche del linguaggio sono quelle che lo rendono di facile apprendimento e che ne hanno consentito una così larga diffusione.
Nella memoria del calcolatore è anche sempre presente il Sistema Operativo, che occupa altri 8K di ROM, le cui routine vengono continuamente usate per procedere nello svolgimento del programma Basic. Le routine del Sistema Operativo sono chiamate KERNAL.
Il programma Basic è formato da una sequenza di frasi o linee. Ogni linea di programma, ad esecuzione differita, inizia con un numero di linea che può andare da 0 a 63999.
Una linea di programma può essere formata da più di una istruzione usando come carattere separatore tra due istruzioni il carattere ":" (due punti). Una linea di programma non può superare 80 caratteri.
Ogni istruzione inizia con una parola chiave, salvo una istruzione, quella di assegnazione, per la quale la parola chiave LET è facoltativa. In generale ogni parola chiave corrisponde a un comando, salvo pochi casi nei quali un comando è formato da più di una parola chiave.

I comandi a volte devono essere accompagnati da alcuni parametri che ne precisano il significato.
Alcuni comandi si servono di operatori, che possono essere di tipo aritmetico, relazionale e logico.

I comandi Basic agiscono su dati che possono essere:

.. 1) costanti;
.. 2) variabili;
.. 3) numeri di linea;
.. 4) indirizzi di memoria.

Riassumendo, le linee di programma comprendono i seguenti elementi:

.. numero di linea;
.. parole chiave;
.. costanti;
.. variabili;
.. operatori;
.. caratteri separatori.

I caratteri separatori ammessi sono:

.. lo spazio;
.. la virgola;
.. il punto e virgola;
.. i due punti.

Possiamo dividere i comandi nelle seguenti categorie:

.. 1) Ingresso/Uscita dati
.. 2) Ingresso/Uscita programmi
.. 3) Trattamento interno dati
.. 4) Controllo
.. 5) Servizio
.. 6) Funzioni.

Nel gruppo 6) delle funzioni possiamo fare le seguenti ulteriori suddivisioni:

.. 6.1) Funzioni matematiche
.. 6.2) Funzioni stringa
.. 6.3) Funzioni conversione formato
.. 6.4) Funzioni controllo

.. 6.5) Funzioni lettura memoria
.. 6.6) Funzioni stampa.

Nei prossimi paragrafi, dopo aver parlato delle variabili, delle costanti e degli operatori, ci soffermiamo sulle caratteristiche di ogni istruzione del linguaggio.

L'alfabeto del linguaggio è composto dai seguenti caratteri:
.. le 26 lettere dell'alfabeto
.. le 10 cifre decimali
.. lo spazio
.. 20 caratteri speciali

riportati, salvo lo spazio, nella tabellina che segue.

```
CARATTERI DEL BASIC

A B C D E F G H I J
K L M N O P Q R S T
U V W X Y Z
0 1 2 3 4 5 6 7 8 9
" # $ % ' ( ) * + , -
/ : ; < = > ? @ ↑
```

La tabellina su esposta è stata ottenuta su stampante dal programma che segue; potrai tornare ad esaminarlo quando avrai imparato il linguaggio Basic.

```
1 REM TABCARATTERI
10 OPEN4,4:CMD4
15 PRINTCHR$(14)"CARATTERI DEL BASIC"
20 PRINT:PRINT
30 K1=65:K2=74:GOSUB100:PRINT
33 K1=75:K2=84:GOSUB100:PRINT
35 K1=85:K2=90:GOSUB100:PRINT
40 K1=48:K2=57:GOSUB100:PRINT
50 K1=34:K2=37:GOSUB100
55 K1=39:K2=45:GOSUB100:PRINT
57 PRINTCHR$(47);" ";
60 K1=58:K2=64:GOSUB100
65 PRINTCHR$(94)CHR$(15)
70 PRINT#4:CLOSE4:STOP
100 FORK=K1TOK2
105 PRINTCHR$(K);" ";:NEXTK:RETURN
```

All'interno delle stringhe sono ammessi tutti i caratteri stampabili e alcuni caratteri di controllo.

2.2 COSTANTI E VARIABILI

Le COSTANTI che possono comparire nelle frasi Basic possono essere:

.. Numeri interi, compresi tra -32768 e + 32767.

.. Numeri reali, che vengono stampati in forma decimale fino a 9 cifre, per passare poi alla notazione esponenziale (floating point), vedi il Capitolo 7 per gli ordini di grandezza.

.. Stringhe di caratteri alfanumerici, delimitate dal carattere virgolette (apici), che non può far parte della stringa. La lunghezza della costante stringa che fa parte della linea di programma deve essere tale da poter essere contenuta negli 80 caratteri consentiti per la linea. Le stringhe non possono comunque superare la lunghezza di 255 caratteri.
Le costanti sono incorporate nelle linee del programma e non sono richiamabili in altri punti del programma. Questo significa che l'uso di costanti può portare a spreco di memoria. Per esempio, se in una istruzione si usa la costante numerica 1984, e in altre 5 istruzioni viene usata ancora la stessa costante numerica 1984, ogni volta vengono usati 4 byte per contenere i 4 caratteri: 1, 9, 8, 4. In tale caso è meglio assegnare un nome simbolico alla costante 1984, che entra così a far parte della categoria delle variabili, e nelle 6 istruzioni richiamare il numero con il nome simbolico della variabile.

Nella memoria del calcolatore sono contenute due costanti speciali; una è richiamabile con il tasto che reca sul fronte il simbolo π e questo risulta il suo nome, l'altra si ottiene usando la funzione EXP(1) ed è il numero irrazionale "e", base dei logaritmi naturali.

Le variabili possono contenere gli stessi tre tipi di dati visti per le costanti; si hanno:

.. Variabili numeriche intere;
.. Variabili numeriche reali;
.. Variabili stringa.

Inoltre si hanno le variabili logiche che non vengono individuate da un nome specifico, il cui valore è assegnabile a variabili numeriche o può fungere da costante numerica nel calcolo di una espressione.

I tre tipi di variabili sono facilmente distinguibili in base al nome che viene loro

assegnato. I NOMI DELLE VARIABILI sono scelti dal programmatore, rispettando le seguenti regole:

.. il nome può essere lungo a piacere, ma vengono usati solo i primi due caratteri per distinguere tra loro le variabili, per questo due variabili diverse non devono mai cominciare con gli stessi due caratteri;

.. il nome di una variabile non deve contenere alcuna sequenza di caratteri che sia una parola chiave del Basic;

.. il nome di una variabile intera viene completato da un suffisso, il carattere %;

.. il nome di una variabile stringa viene completato da un suffisso, il carattere $;

.. il nome di una variabile reale termina senza suffisso;

.. i caratteri validi per formare il nome di una variabile sono le 26 lettere dell'alfabeto e le 10 cifre decimali, però il primo carattere del nome deve essere una lettera.

Esempi di nomi di variabili sono:

.. per numeri interi:

A1% AC% B% PLUTO% (che viene riconosciuto come PL%);

.. per numeri reali:

A1 BD CASA (che viene riconosciuto come CA);

.. per stringhe:

A1$ C$ W$ BELLO$ (che viene riconosciuto come BE$).

Non si può scrivere:

.. TOPO%, infatti viene riconosciuto come TO% e TO è una parola chiave;

.. GOLF, infatti contiene la parola chiave GO;

.. TIPO e TICCHIO vengono riconosciute come la variabile TI.

Si può usare: TA% insieme a TA e insieme a TA$, si tratta infatti di tre variabili diverse.

Qualora in un programma venga richiamata una variabile, alla quale non è ancora stato assegnato un valore, il sistema considera il valore zero per la variabile numerica e il valore di stringa nulla per la variabile stringa. La stringa nulla è una stringa di lunghezza zero, cioè formata da nessun carattere, cosa molto diversa dalla stringa formata da un carattere spazio.

Le due stringhe: A$=" " (uno spazio)
B$="" (stringa nulla)

sono diverse, la prima ha lunghezza 1 e contiene il carattere "spazio", la seconda ha lunghezza 0 e non contiene caratteri.

Una stringa può contenere qualunque carattere salvo le virgolette; infatti le virgolette delimitano le stringhe. Per poter fare entrare le virgolette in una stringa bisogna passarle con la funzione CHR$ usando come argomento il codice ASCII 34.

Le variabili vengono riempite, cioè contengono dei dati, per effetto di una delle seguenti operazioni:

.. assegnazione di un dato. Si ottiene scrivendo il nome della variabile seguito dal carattere "=" (uguale) e dal dato; esempi:

A$ ="pippo" A%=897 A=4567.98

In questa istruzione di assegnazione si può usare la parola chiave LET prima del nome della variabile.

.. lettura di un dato. Si ottiene scrivendo il nome della variabile dopo la parola chiave che indica un'operazione di lettura. L'operazione di lettura può avvenire dall'esterno con la parola chiave INPUT o dall'interno del programma con la parola chiave READ.

Le variabili considerate fino ad ora sono VARIABILI SINGOLE, cioè un nome corrisponde a un dato.

Esistono le VARIABILI DI GRUPPO; esse vengono abitualmente chiamate VARIABILI CON INDICE. Un solo nome definisce un gruppo di variabili; all'interno del gruppo si usano degli indici numerici per distinguere le singole variabili. Esse vengono spesso chiamate ARRAY o MATRICI.
In molti casi si trattano dati che appartengono, per una qualche ragione, ad un gruppo. Una variabile con indice può avere una o più dimensioni, cioè per individuarla può essere sufficiente un solo indice, o possono servirne più di uno. In questa implementazione del Basic non esiste un limite teorico al numero delle dimensioni

di una variabile con indice. Per numero delle dimensioni intendiamo il numero di indici assegnabili.
E' immediato paragonare una variabile con un solo indice ad una lista di variabili. Analogamente, pensando a una variabile con due indici, viene subito in mente una tabella, con un certo numero di righe e di colonne, nella quale ogni variabile rappresenta un nodo, cioè il punto di incontro tra una riga e una colonna. Continuando, una variabile con tre indici può far venire in mente un parallelepipedo; ogni variabile si trova su una riga e su una colonna di un piano, cioè a una certa quota.

Si può continuare, ma la geometria elementare non ci viene più facilmente in aiuto. Non devi confondere il numero delle dimensioni con l'ordine di grandezza di ogni dimensione.

Una lista di 100 variabili può esser pensata come una variabile a una dimensione formata da 100 elementi. Una variabile relativa a 20 argomenti, ognuno formato da 3 dati, viene immaginata come una tabella formata da 20 righe di 3 colonne ciascuna; sono in tutto 60 elementi, la dimensione riga varia da 1 a 20, mentre la dimensione colonna varia da 1 a 3.
Il numero degli elementi in ogni dimensione può raggiungere il valore massimo per gli interi positivi, cioè 32767. Il numero delle dimensioni può raggiungere 255. Ma attenzione, 255x32767 supera il numero dei byte della memoria del nostro calcolatore!

Per la definizione delle variabili con indice esiste una istruzione apposita: DIM (DIMension). Questa istruzione deve essere obbligatoriamente usata per poter trattare variabili con indice per le quali almeno una dimensione superi gli 11 elementi. Qualora si stia al disotto di questo valore, non è necessario usare la istruzione DIM.

Le variabili con indice possono essere dei tre tipi già trattati: intere, reali e stringhe. Per la formazione dei nomi valgono le regole già viste; il nome delle variabili viene seguito da una coppia di parentesi, all'interno delle quali si scrivono gli indici, separati da virgole, se sono più di uno.
Dopo aver eseguito l'operazione di dimensionamento, il sistema assegna il valore 0 alle variabili numeriche e il valore stringa nulla alle variabili stringa.

Esempi:

DIM A(24) crea la variabile con indice A, con un solo indice (tra le parentesi compare un solo numero), formata da 25 elementi numerici reali e assegna il valore zero ad ogni elemento. Il primo elemento è A(0) e l'ultimo è A(24). Gli indici partono dal valore zero.

DIM BR$(15) crea la variabile con indice BR$, formata da 16 stringhe, tutte al valore iniziale di stringa nulla.

Se nel programma si cita la variabile B%(5), che non è stata dimensionata, il sistema fa un dimensionamento implicito, come se si fosse scritto DIM B%(10), cioè crea una variabile con indice formata da 11 elementi interi con valore iniziale uguale a zero.

Nel Capitolo 8 descriviamo in dettaglio l'occupazione di memoria per ogni tipo di variabile.

Le VARIABILI LOGICHE sono variabili che possono assumere due soli valori: VERO e FALSO. In realtà esse possono essere considerate variabili numeriche. Nel prossimo paragrafo trattiamo ampiamente l'argomento.

2.3 OPERATORI ARITMETICI, RELAZIONALI E LOGICI

Il Basic accetta ESPRESSIONI formate da COSTANTI e VARIABILI collegate tra loro da OPERATORI ARITMETICI e/o RELAZIONALI e/o LOGICI. Nella spiegazione dettagliata delle istruzioni del linguaggio, nel Paragrafo 2.4, abbiamo sempre messo in rilievo quando si possono usare espressioni.

Gli OPERATORI ARITMETICI accettati sono:

Operatori	*Significato*
+	somma
—	sottrazione
*	moltiplicazione
/	divisione
↑	elevamento a potenza
=	uguale a
(	aperta parentesi
)	chiusa parentesi

con i normali significati dell'aritmetica.

Gli OPERATORI RELAZIONALI accettati sono:

Operatori	*Significato*
<	minore di
=	uguale a
>	maggiore di
<=	minore di o uguale a
>=	maggiore di o uguale a
<>	non uguale a

e servono a stabilire una relazione sia in senso artitmetico che non aritmetico.

Gli OPERATORI LOGICI accettati sono:

Operatori	*Significato*
AND	e (l'uno e l'altro)
OR	o (uno o l'altro)
NOT	no (negazione)

e possono collegare tra loro sia espressioni aritmetiche che relazionali.

Le espressioni vengono elaborate partendo da sinistra e procedendo verso destra, rispettando le regole di precedenza che seguono (riportate dalla più alta alla più bassa), ma tenendo presente che le espressioni racchiuse in parentesi hanno a loro volta la precedenza.

REGOLE DI PRECEDENZA

↑	elevamento a potenza
-	negazione (segno meno)
* /	moltiplicazione e divisione
+ -	somma e sottrazione
> = <	relazione
NOT	NO logico
AND	AND logico
OR	OR logico

Nell'elaborazione delle espressione aritmetiche vengono applicate integralmente le regole dell'algebra; devi fare attenzione al tipo delle variabili aritmetiche che sono coinvolte e ricordare queste regole:

.. ogni operazione avviene tra due operandi;
.. sia che i due operandi siano interi, sia che siano uno intero e l'altro reale, sia che siano tutti e due reali il risultato è reale;
.. alla fine del calcolo il valore viene assegnato alla variabile che sta a sinistra dell'uguale rispettandone il tipo.

Per poter ottenere risultati interi, senza doverli necessariamente assegnare ad una variabile di tipo intero, si può usare la funzione INT, che fornisce la parte intera di un risultato.

L'unica operazione aritmetica eseguibile tra stringhe è la somma (+), nel senso che sommare due stringhe significa scriverle una di seguito all'altra, ottenendo una stringa più lunga (massimo 255 caratteri).

Gli operatori relazionali si usano per confrontare tra loro due operandi; questo confronto ha il normale significato aritmetico per due operandi numerici, mentre ha un significato basato sull'ordinamento per gli operandi di tipo stringa. Le stringhe sono formate da caratteri ASCII; ogni carattere corrisponde a un codice numerico. Quando si confrontano tra loro due stringhe, esse sono confrontate carattere per carattere in base al valore numerico del codice. In tale confronto viene rispettato l'ordine alfabetico normale. Se si confronta una stringa, con una più lunga, ma che ha i primi caratteri uguali, la più lunga risulta maggiore.

"ABBA" risulta minore di "ABBAGLIO".

Puoi vedere nel Capitolo 7 i valori dei codici ASCII nei due set di caratteri.

Nel confronto tra due operandi, di qualunque tipo, nasce una VARIABILE LOGICA, che, da un punto di vista logico ha il significato di VERO o FALSO, ma in realtà ha un valore aritmetico: il numero intero -1 per VERO e il numero intero 0 per FALSO. Per questa ragione in espressioni di tipo aritmetico può essere compresa una espressione relazionale; ad essa, durante il calcolo, viene sostituito il numero intero risultato. Ha quindi senso scrivere:

A%=C$>B$ A% contiene -1 se C$ è maggiore di B$
A% contiene 0 se C$=B$ o se C$<B$.

Si possono operare confronti tra numeri interi e numeri reali; per esempio:

A=1237 e A%=1237 se confrontati risultano uguali.

Devi fare particolare attenzione ai confronti tra numeri che risultano da calcoli complessi, dato che, anche se quando stampi due numeri essi sono uguali, in memoria possono essere leggermente diversi; sono scherzi dovuti al calcolo binario. Senza andare in calcoli troppo complicati, considera il programma CALCOLI che segue, e i suoi rusultati; ti aspetteresti che A e B siano uguali, mentre non è così.

```
1 REM CALCOLI
10 OPEN4,4:CMD4
20 A=(3↑(-10))
30 B=(1/3)↑10
40 PRINT"PRIMO CALCOLO: (3↑(-10)) =";A
50 PRINT"SECONDO CALCOLO: (1/3)↑10=";B
60 PRINT"DIFFERENZA CALCOLATA:      =";A-B
70 PRINT#4:CLOSE4:STOP
```

RISULTATI PROGRAMMA CALCOLI

```
PRIMO CALCOLO: (3↑(-10)) = 1.69350878E-05
SECONDO CALCOLO: (1/3)↑10= 1.69350879E-05
DIFFERENZA CALCOLATA:     =-1.20792265E-13
```

Gli OPERATORI LOGICI (detti anche BOOLEANI) servono per collegare tra loro due o più espressioni di qualunque tipo. Essi possono essere usati per arricchire il significato di espressioni di tipo relazionale, o per considerare risultati aritmetici diversi in relazione tra loro. L'applicazione di questi operatori produce un risultato logico di VERO o FALSO espresso in modo aritmetico; se il numero assegnato alla variabile logica è 0 (zero) il risultato viene considerato falso in una analisi di condizione, mentre se esso è diverso da zero (non sempre risulta -1) viene considerato vero.

Gli operatori logici lavorano confrontando i bit corrispondenti con i seguenti risultati:

Oper. logica	Risult. aritmetico	Risult. relazionale
1 AND 1	1	VERO
1 AND 0	0	FALSO
0 AND 1	0	FALSO
0 AND 0	0	FALSO
1 OR 1	1	VERO
1 OR 0	1	VERO
0 OR 1	1	VERO
0 OR 0	0	FALSO
NOT 1	0	FALSO
NOT 0	1	VERO

Nel programma OPERATORI LOGICI che segue si stampano 16 esempi per capire come lavorano gli operatori logici.

```
1 REM OPERATORI LOGICI
3 OPEN4,4:CMD4
5 PRINT"ESEMPI USO OPERATORI LOGICI":PRINT
10 A=27:B=90:C=45:D=4
20 PRINT" 1) ";"A<B AND D<C   ";A<BANDD<C
25 X=A<BANDD<C
30 PRINT" 2) ";"X=A<B AND D<C =";X
```

```
40 PRINT" 3) ";"A>B AND D<C : ";A>BANDD<C
45 Y=A>BANDD<C
50 PRINT" 4) ";"Y=A>BANDD<C =";Y
60 Z=AANDB
65 PRINT" 5) ";"Z=A AND B : ";Z
70 PRINT" 6) ";"A<B OR D<C : ";A<BORD<C
73 X=A<BORD<C
75 PRINT" 7) ";"X=A<B OR D<C =";X
80 PRINT" 8) ";"A>B OR D<C : ";A>BORD<C
83 Y=A>BORD<C
85 PRINT" 9) ";"Y=A>BORD<C =";Y
87 Z=AORB
89 PRINT"10) ";"Z=A OR B : ";Z
90 PRINT"11) ";"NOT B : ";NOT B
93 IFNOTA>BTHENPRINT"12) ";"NOT A>B; VERO: A<B"
95 PRINT"13) ";"NOT A > B : ";NOTA>B
97 IFNOTB<ATHENPRINT"14) ";"NOT B<A; VERO: A<B"
99 PRINT"15) ";"NOT A < B : ";NOTA<B
100 Z=NOTA
105 PRINT"16) ";"A=";A,"NOT A=";Z
110 PRINT#4:CLOSE4:STOP
```

RISULTATI PROGRAMMA OPERATORI LOGICI

```
ESEMPI USO OPERATORI LOGICI

  1) A<B AND D<C : -1
  2) X=A<B AND D<C =-1
  3) A>B AND D<C :  0
  4) Y=A>BANDD<C = 0
  5) Z=A AND B :  26
  6) A<B OR D<C : -1
  7) X=A<B OR D<C =-1
  8) A>B OR D<C : -1
  9) Y=A>BORD<C =-1
 10) Z=A OR B :  91
 11) NOT B : -91
 12) NOT A>B; VERO: A<B
 13) NOT A > B : -1
 14) NOT B<A; VERO: A<B
 15) NOT A < B :  0
 16) A= 27              NOT A=-28
```

All'inizio del programma vengono assegnati quattro valori numerici alle quattro variabili: A, B, C, e D. Dopo si mandano alla stampante sedici risultati che commentiamo qui.

1) A<B AND D<C fornisce il risultato aritmetico -1 nel senso di relazione VERA, infatti : 27<90 e 4<45.

2) Assegnando alla variabile X il risultato della precedente relazione composta si ottiene -1, dato che la relazione è VERA.

3) A>B AND D<C fornisce il risultato aritmetico 0 nel senso di relazione FALSA.

4) Assegnando alla variabile Y il risultato della precedente relazione composta si ottiene 0, dato che la relazione è FALSA.

5) Z = A AND B dà il risultato 26 perchè, considerando i valori binari dei numeri si ha:

A=27	in binario	A=00011011
B=90	in binario	B=01011010
A AND B	in binario	Z=00011010 Z=26

infatti per effetto della operazione AND si ha un bit 1 dove tutti e due gli operandi hanno 1.

6) A<B OR D<C fornisce il risultato aritmetico -1 nel senso di relazione VERA. In particolare viene eseguito OR di due relazioni vere, per dare risultato -1 basta che sia vera una delle due.

7) Sempre per la stessa ragione del punto 6) X riceve il valore -1.

8) In questo caso si ha OR di due relazioni, una vera e una falsa, e il risultato è VERO.

9) Vale quanto detto in 8).

10) Si opera OR tra A e B, con risultato 91; considerando i valori binari si ha:

A=27	in binario	A=00011011
B=90	in binario	B=01011010
A OR B	in binario	Z=01011011 Z=91

infatti si ottiene un bit 1 dove anche uno solo dei due operandi ha 1.

11) NOT B =—91, infatti B=90, consideriamo i valori binari:

    B=90 in binario B=01011010 e operando con NOT su ogni bit si ottiene:

    NOT B=10100101 che interpretato come numero, dato che inizia con un bit 1, dà il valore negativo —91.

12) NOT A>B risulta VERO, infatti la relazione A>B è FALSA; l'operatore NOT viene applicato alla relazione. Se si prova, per gli stessi valori di A e B:

    PRINT (NOTA)>B si ottiene 0, infatti in questo caso NOT viene applicato ad A e NOT A=—28 risulta <90, e quindi la relazione è FALSA.

13) Vale quanto detto all'inizio del punto 12).

14) L'operatore NOT si applica ad una relazione falsa e quindi dà un risultato VERO.

15) L'operatore NOT si applica ad una relazione vera e quindi dà un risultato FALSO, uguale 0 aritmeticamente.

16) Per quanto visto già al punto 11), essendo A=27, NOT A=—28. Puoi provare a rifare i calcoli in binario.

Ci rendiamo conto che i programmi esempio riportati compaiono troppo presto; non sono infatti ancora state spiegate tutte le istruzioni. Se fai fatica a seguire i programmi, potrai riprenderli in esame più tardi.

Seguono ulteriori spiegazioni sugli operatori logici, corredate da esempi.

AND   Tipo: Operatore logico

Formato: espressione AND espressione

Abbiamo già messo in evidenza, sia l'aspetto aritmetico, che quello di controllo tra due relazioni, di questo operatore logico.

Puoi provare il programma che segue per diversi valori dei due numeri A e B e riflettere sui risultati.

```
1 REM ES2 PROVA AND
5 A$="A>B AND B<>0"
10 INPUT"PRIMO: ";A
20 INPUT"SECONDO: ";B
30 PRINT"A AND B";AANDB
40 PRINT"A AND NOT B";AANDNOTB
50 PRINT"NOT A AND B";NOTAANDB
60 IFA>BANDB<>0THENPRINTA$:GOTO80
70 PRINT"NON VERIFICATO"
80 STOP
```

Dato che l'operatore agisce su numeri compresi tra -32768 e +32767, se dai per A e/o B valori fuori dai limiti, avrai il messaggio di errore: ?ILLEGAL QUANTITY.

NOT Tipo: Operatore logico

Formato: NOT espressione

dove espressione può essere una qualunque espressione numerica.

Si tratta di un operatore logico che può essere usato in modo aritmetico o logico. Se l'espressione numerica dà un risultato reale, l'operatore lavora solo sulla parte intera, che deve essere compresa tra —32768 e +32767. In caso contrario si ha il messaggio: ?ILLEGAL QUANTITY. Quando lavora in senso aritmetico su un numero N, il numero NOT N è: -(N+1).

Segue un programma che ti consente di riflettere sul significato di questo operatore.

```
1 REM ES36 PROVA NOT
5 R$="NOT(N1<N2): "
10 INPUT"NUMERO: ";N
20 PRINT"N=";N;"NOT N=";NOTN
30 INPUT"SCRIVI 2 NUMERI: ";N1,N2
40 PRINT"I DUE NUMERI SONO: ";N1;" , ";N2
50 IFN1>N2THENPRINT"N1>N2":GOTO60
55 PRINT"N1<=N2"
60 IFNOTN1<N2THENPRINT"NOTN1<N2: ";NOTN1<N2
63 PRINT"NOTN1: ";NOTN1;"N2: "N2
65 IFNOT(N1<N2)THENPRINTR$;NOT(N1<N2)
67 PRINT"N1<N2: ";N1<N2
70 STOP
```

OR Tipo: Operatore logico

Formato: espressione OR espressione

dove espressione può essere una relazione logica o una espressione numerica.

Questo operatore logico può essere usato per correlare tra loro due relazioni e quindi può intervenire in una scelta tra vero e falso. Nell'uso numerico valgono le regole viste.

L'operatore lavora su numeri interi compresi tra -32768 e +32767, con operandi fuori da questi limiti si ha segnalazione di errore.

Segue un programma che esemplifica quanto detto.

```
1 REM ES39 PROVA OR
10 INPUT"SCRIVI 2 NUMERI";N1,N2
20 PRINT"N1=";N1;" N2=";N2
30 PRINT"N1 OR N2 = ";N1ORN2
40 IFN1<N2ORN1<300THENPRINT"N1<N2 OR N1<300"
50 IFN1<0ORN2<0THENPRINT"ALMENO UNO NEGATIVO"
```

2.4 ISTRUZIONI

In questo paragrafo elenchiamo le istruzioni del linguaggio in ordine alfabetico, onde consentirne una facile ricerca in caso di dubbi.

Ricordiamo che le linee di un programma, che deve essere eseguito in modo differito, si scrivono numerandole progressivamente. E' buona regola non usare numeri consecutivi; questo facilita eventuali inserzioni nella fase di prova del programma.
Una linea di programma può contenere più istruzioni; in tale caso devi usare come carattere separatore tra le istruzioni il carattere ":" (due punti).
Tutte le istruzioni, salvo quella di assegnazione, iniziano con una parola chiave. Per l'istruzione di assegnazione la parola chiave LET è facoltativa.
Una linea di programma non può superare gli 80 caratteri, 2 righe del video. In realtà si riescono a superare gli 80 caratteri, che costituiscono un limite nella fase della scrittura del programma, usando le abbreviazioni consentite per le parole chiave (vedi Capitolo 7). Questo però presenta un rischio; infatti quando si chiede di listare sul video il programma, sono listate anche le istruzioni che, riportando in modo esteso le parole chiave, superano gli 80 caratteri, però, se si apporta qualche correzione alla linea, quando si preme RETURN, la linea viene troncata. Per

evitare questo inconveniente si dovrebbe tornare a contrarre le parole chiave per far entrare di nuovo la linea di programma in 80 caratteri. Questo modo di procedere risulta un poco macchinoso. Ti consigliamo pertanto di non scrivere linee di programma troppo lunghe.

Alcune istruzioni non possono essere usate in modo immediato, mentre per la maggior parte questo è possibile. Nel seguito segnaliamo esplicitamente le istruzioni che non possono essere usate in modo immediato e quelle che non ha senso usare in modo differito, anche se è consentito farlo.

Nel Paragrafo 2.1 abbiamo definito sei categorie per le istruzioni Basic; ora elenchiamo le istruzioni che appartengono ad ogni gruppo.

1) Ingresso e Uscita dati, abbreviato in Operazioni I/O, da Input/Output:

CLOSE	INPUT#
CMD	OPEN
GET	PRINT
GET#	PRINT#
INPUT	

2) Operazioni I/O programmi:

LOAD	VERIFY
SAVE	

3) Trattamento interno dati:

DATA	POKE
DIM	READ
LET	RESTORE

4) Controllo:

CONT	ON
END	RETURN
FOR.. TO.. STEP	RUN
GOSUB	STOP
GOTO	SYS
IF.. THEN	WAIT
NEXT	

5) Servizio:

CLR	NEW
DEF FN	REM
LIST	

6) Funzioni così ripartite:

6.1) Matematiche	6.2) Stringa
ABS	CHR$
ATN	LEFT$
COS	MID$
EXP	RIGHT$
FN	STR$
INT	
LOG	
RND	
SGN	
SIN	
SQR	
TAN	

6.3) Conversione	6.4) Controllo
ASC	USR
VAL	

6.5) Lett. memoria	6.6) Stampa
FRE	SPC
LEN	TAB
PEEK	
POS	
STATUS	
TIME	
TIME$	

Prima di passare alla descrizione delle istruzioni preferiamo giustificare i raggruppamenti.

Abbiamo definito il gruppo 1 per le istruzioni che riguardano l'entrata e l'uscita dei dati rispetto al calcolatore. Il gruppo 2 è stato invece definito per l'entrata e l'uscita dei programmi. Il gruppo 3 riguarda quelle operazioni che modificano i contenuti della memoria, vuoi per semplici trasferimenti che per calcoli eseguiti. Nel gruppo 4 rientrano tutte quelle istruzioni che controllano lo svolgimento del programma. Nel gruppo 5 abbiamo lasciato poche istruzioni di servizio. Nel gruppo 6 abbiamo raggruppato tutte le funzioni, procedendo poi ad altre 6 suddivisioni in base al tipo delle funzioni stesse. Per esempio in 6.4) entra la funzione USR che manda in esecuzione un programma in linguaggio macchina e quindi agisce come la SYS del gruppo 4. La CMD, che abbiamo messo nel gruppo 1, in realtà potrebbe anche essere messa altrove, per esempio nel gruppo 5, tra le istruzioni di servizio. Qualcuno potrebbe ritenere più appropriato porre OPEN e CLOSE ancora nel gruppo 5.

Si potrebbe discutere a lungo su queste suddivisioni; anche noi non siamo sicuri di avere fatto buone scelte. Ti invitiamo pertanto ad entrare nella discussione, quando sarai diventato un esperto del linguaggio Basic. Una proposta potrebbe essere che, essendo il Basic un linguaggio di tipo interpretativo, qualunque istruzione venga considerata una funzione svolta servendosi di uno o più sottoprogrammi.

Nel seguito, nella descrizione delle istruzioni, si usano le seguenti convenzioni:

.. per ogni istruzione si scrivono la o le parole chiave, in stampatello;
.. si cita il gruppo di appartenenza;
.. si espone il formato di scrittura;
.. gli operandi e i parametri sono scritti in caratteri minuscoli;
.. le parti opzionali sono racchiuse in parentesi quadre;
.. i puntini significano ripetizione;
.. per i parametri e gli operandi usiamo alcune abbreviazioni:

lfn	numero-logico-file
var	variabile
fn	nome-file
dn	numero-logico-periferica (device number)
sa	indirizzo-secondario
ft	tipo-file
md	modo

.. espressione significa una qualunque espressione, al limite una sola variabile o costante.

Riportiamo la tabellina delle dn relative alle più comuni periferiche.

PERIF.	DN	SA
Cassette	1	0 = Input 1 = Output 2 = Output con EOT
Video	3	
Stampante	4 o 5	0 = Set maiuscolo/grafico 7 = Set maiuscolo/minuscolo
Disco	8/11	0/1 = trasferimento programmi 2/14 = canali dati 15 = canale comandi

ABS Tipo: Funzione matematica

Formato: ABS (espressione)

dove "espressione" deve dar luogo ad un valore numerico.

La funzione ABS fornisce il valore assoluto dell'argomento, cioè l'argomento stesso, se esso è positivo, l'argomento moltiplicato per -1, se esso è negativo. Puoi provare il programma che segue; ti viene chiesto un numero e sono stampati sul video il numero e il suo valore assoluto.

```
1 REM ES1 PROVA ABS
10 INPUT"SCRIVI UN NUMERO ";X
15 A$=" VALORE ASSOLUTO DI X="
20 PRINT"X=";X;A$;ABS(X)
```

ASC Tipo: Funzione conversione formato

Formato: ASC (stringa)

dove stringa può essere sia una stringa delimitata dalle virgolette, cioè una costante, che una variabile stringa.

Il risultato fornito da questa funzione è un numero compreso tra 0 e 255, tale numero è il codice ASCII del primo carattere della stringa argomento. Si può avere il messaggio di errore: ?ILLEGAL QUANTITY, se la stringa argomento è la stringa nulla.

Puoi provare il programma che segue, rispondendo alla richiesta con diverse stringhe; puoi anche rispondere con la stringa nulla; infatti alla linea 20 abbiamo usato un accorgimento aggiungendo alla stringa A$ la stringa di un carattere CHR$(0). Puoi verificare sulla tabella dei codici ASCII il codice numerico stampato.

```
1 REM ES3 PROVA ASC
10 INPUT A$
20 PRINTASC(A$+CHR$(0))
30 STOP
```

ATN Tipo: Funzione matematica

Formato: ATN (numero)

dove numero può anche essere una variabile numerica o una espressione numerica.

Il risultato è l'angolo, espresso in radianti, che ha come tangente l'argomento fornito. Puoi provare il programma che segue per diversi valori dell'argomento.

```
1 REM ES4 PROVA ATN
2 INPUT"ARGOMENTO: ";A
3 OPEN4,4:CMD4
4 A$="ATN(":C$="GRADI: "
5 B$=")=":D$="RAD.: "
6 PRINTD$A$;"1";B$;ATN(1)
7 PRINTC$A$;"1";B$;ATN(1)*180/π
8 PRINTD$A$;"0";B$;ATN(0)
9 PRINTC$A$;"0";B$;ATN(0)*180/π
20 PRINTD$A$;A;B$;ATN(A)
25 PRINTC$A$;A;B$;ATN(A)*180/π
30 PRINT#4:CLOSE4:STOP
```

Ecco alcuni risultati del programma:

```
RAD.: ATN(1)= .785398163
GRADI: ATN(1)= 45
RAD.: ATN(0)= 0
GRADI: ATN(0)= 0
RAD.: ATN( 38 )= 1.54448661
GRADI: ATN( 38 )= 88.4925643
```

Come vedi alle linee 6, 8 e 25 si trasforma il risultato in gradi. Il risultato fornito dalla funzione è sempre compreso tra $-\pi/2$ (-1.57079633) e $+\pi/2$ (+1.57079633).

CHR$ Tipo: Funzione stringa

Formato: CHR$ (numero)

dove numero può anche essere una variabile o un'espressione numerica, ma l'argomento deve essere compreso tra 0 e 255 (valori possibili per 1 byte). La funzione fornisce una stringa di un carattere, il carattere che corrisponde al codice. Se il codice si riferisce a un carattere non stampabile la stringa prodotta ha lunghezza 1, ma stampandola non compare alcunchè. La funzione contraria di CHR$ e ASC.

Puoi provare il programma che segue e vedrai confermato quanto abbiamo detto.

```
1 REM ES5 PROVA CHR$
10 PRINT"CHR$(0)=";CHR$(0)
20 PRINT"CHR$(255)=";CHR$(255)
30 PRINT"CHR$(127.67)=";CHR$(127.67)
40 PRINT"CHR$(127)=";CHR$(127)
43 A=56:A$=CHR$(A):B=ASC(A$)
45 PRINT"CHR$(A)=";CHR$(A)
47 PRINT"ASC(A$)=";ASC(A$)
50 INPUT"CODICE ASCII ";A
60 PRINT"CHR$(";A;")=";CHR$(A)
70 STOP
```

RISULTATI PROGRAMMA ES5

```
CHR$(0)=
CHR$(255)=π
CHR$(127.67)=◥
CHR$(127)=◥
CHR$(A)=8
ASC(A$)= 56
CODICE ASCII CHR$( 48 )=0
```

Se l'argomento di CHR$ è >255 si ha il messaggio di errore: ?ILLEGAL QUANTITY, mentre se esso è un numero con decimali, ma nei limiti consentiti, viene presa la parte intera e non si ha segnalazione di errore.

CLOSE Tipo: Operazione di I/O dati

Formato: CLOSE lfn

dove lfn, che può essere una costante o una variabile, è il numero logico del file usato nella operazione di OPEN che ha consentito di lavorare con quel file usando un certo canale. Fai attenzione che nella CLOSE si usa il numero logico del file e non il numero del canale. Pensiamo sia necessario puntualizzare questo fatto, dato che spesso si ha l'abitudine di usare gli stessi numeri per canale e numero logico file. L'operazione CLOSE è molto importante perchè quando si lavora con unità di uscita come cassette o floppy, che usano un BUFFER per ammucchiare i dati da scrivere, è proprio essa che va a scrivere il contenuto dell'ultimo blocco e registra la chiusura del file. Un file non chiuso, non può poi essere riletto. Questa operazione va comunque sempre eseguita a conclusione di una operazione iniziata con una OPEN; infatti nella memoria di lavoro del calcolatore, per ogni OPEN viene memorizzato un insieme di informazioni che continua a sussistere fino alla CLOSE corrispondente ed è dannoso tenere occupata la relativa tabella con informazioni

che non servono più.
Nel programma che segue con un ciclo si chiudono tutti i file con numero-logico da K1 a K2.

```
1 REM ES6 PROVA CLOSE
10 REM CHIUSURA DI TUTTI I FILE
20 FORK=K1TOK2
30 CLOSEK
40 NEXTK
```

CLR Tipo: Servizio

Formato: CLR

Questa istruzione PULISCE tutte le variabili del programma, cioè pone zero in quelle numeriche e stringa-nulla in quelle stringa. Inoltre annulla tutte le memorizzazioni relative ad operazioni in corso come: GOSUB, cicli FOR...NEXT, funzioni definite dall'utente, OPEN di file. Quest'ultimo effetto può essere pericoloso, perchè eventuali file ancora aperti non vengono chiusi in modo corretto; per il sistema disco, ad esempio, i file sono ancora aperti, mentre per il calcolatore non lo sono più.
Segue un programma dimostrativo e sono riportati i risultati come appaiono sul video.

```
1 REM ES7 PROVA CLR
10 A=34.15:A%=89:A$="PROVA CLR"
20 PRINT"STAMPA VARIABILI":GOSUB50
30 CLR
40 PRINT"STAMPA DOPO CLR":GOSUB50
45 STOP
50 PRINT"A=";A;" A%=";A%;" A$=";A$
55 RETURN
```

RISULTATI PROGRAMMA PRECEDENTE

```
STAMPA VARIABILI
A= 34.15  A%= 89  A$=PROVA CLR

STAMPA DOPO CLR
A= 0  A%= 0  A$=


BREAK IN 45
```

CMD Tipo: Operazione I/O dati

Formato: CMD lfn [,stringa]

dove lfn è il numero logico di un file che deve essere stato preventivamente aperto con OPEN su una periferica diversa dal video. La stringa che segue è opzionale; se presente viene mandata al file e può essere considerata come titolo.
Questa operazione pone la periferica, aperta con il numero-logico di file specificato, in stato di attesa, sostituendola al video. Le successive operazioni di PRINT e LIST agiscono verso la periferica designata. Per far tornare attivo il video si deve scrivere una istruzione PRINT# lfn, senza operandi; questo corrisponde all'invio di una linea bianca alla periferica, e poi si deve chiudere il file con CLOSE. Se si verifica un errore del tipo: ?SYNTAX ERROR l'uscita dei dati torna verso il video, ma la periferica rimane in stato di attesa e quindi si genera una situazione irregolare; devi quindi usare la procedura sopra descritta anche in questo caso.
Segue un programma esempio che dapprima, linee da 5 a 30, manda un file sequenziale su cassetta, chiamandolo PROVA CASSETTA, e poi apre la stampante come file 4, le invia un messaggio e lista se stesso.

```
1 REM ES8 PROVA CMD
5 REM MANDA FILE SEQUENZIALE SU CASSETTA
10 OPEN1,1,1,"PROVA CASSETTA"
20 FORK=1TO26:PRINTCHR$(64+K):NEXTK
30 PRINT#1:CLOSE1
35 OPEN4,4:CMD4
40 PRINT"SCRITTO FILE SEQ. SU CASSETTA"
45 PRINT"LISTA PROGRAMMA"
50 LIST
```

RISULTATI PROGRAMMA PRECEDENTE

```
SCRITTO FILE SEQ. SU CASSETTA
LISTA PROGRAMMA
```

Dopo l'esecuzione del programma devi in modo immediato scrivere:

PRINT# 4:CLOSE4

per chiudere correttamente, infatti dopo LIST il controllo non torna più al programma in corso.

Fai attenzione al fatto che, se nella lista di stampa sono compresi anche caratteri di controllo per il video, essi possono avere effetti nulli o diversi per altre periferiche.

CONT Tipo: Controllo

Formato: CONT

Si tratta di un comando che non ha senso scrivere come parte di un programma da usare in modo differito. Il suo uso è in modo immediato per far ripartire un programma dal punto dove si è fermato per effetto delle istruzioni STOP o END incontrate o per l'uso estemporaneo del tasto RUN/STOP. Se il programma si ferma per aver incontrato l'istruzione END non compare alcun messaggio. Negli altri due casi compare il messaggio: BREAK IN numero-linea. Scrivendo CONT e premendo RETURN il programma continua.
Devi fare attenzione, quando il programma è fermo, a non svolgere alcuna azione di EDIT; puoi solo chiedere dei LIST totali o parziali, stampare il valore di variabili da controllare, modificare in modo immediato i contenuti di qualche variabile. In tutti questi casi scrivendo CONT il programma procede. Basta che tu, dopo la fermata, chieda un LIST, porti il cursore su una linea e prema RETURN anche senza aver apportato modifiche (operazione di EDIT) perchè al comando CONT venga risposto con il messaggio: CAN'T CONTINUE ERROR.
E' importante imparare ad usare bene CONT, infatti nella fase di messa a punto dei programmi è opportuno piazzare degli STOP in punti strategici (meglio STOP che END per vedere in quale linea si è fermato) e poi andare ad indagare su particolari situazioni delle variabili.
Segue un programma che calcola per approssimazioni successive il valore di π.

```
1 REM ES9 PROVA CONT
10 PI=0:C=1
20 PI=PI+4/C-4/(C+2)
30 PRINT "PI=";PI;"    π=";π
40 C=C+4:GOTO20
```

Puoi provare a farlo girare e premere RUN/STOP, poi scrivere in immediato:

PRINT C e poi RETURN

per vedere a quale valore di C sei e a quale punto dell'approssimazione, del valore di π calcolato da te, sei rispetto a quello della macchina. Per rallentare lo scrolling del video puoi tenere premuto il tasto CTRL.
Se provi vedrai che quando C arriva oltre 3370, nel valore calcolato sono giuste le prime 3 cifre decimali.

COS Tipo: Funzione matematica

Formato: COS (numero)

dove numero può anche essere una variabile o un'espressione numerica. L'argomento deve essere la misura di un angolo espresso in radianti.
Segue un esempio, nel quale, se scegli di introdurre l'angolo in gradi, il programma lo trasforma in radianti prima di calcolare il coseno.

```
1 REM ES10 PROVA COS
10 INPUT"GRADI O RADIANTI (G/R) ";R$
20 INPUT"ARGOMENTO: ";A
30 IFR$="G"THENA=A*π/180
40 PRINT"COS(";A;")=";COS(A)
50 STOP
```

DATA Tipo: Trattamento interno dati

Formato: DATA lista-costanti

dove lista-costanti è una serie di costanti anche di tipo diverso, separate dal carattere separatore virgola (,).
Si tratta di una istruzione che non viene propriamente eseguita, per cui non importa dove è sistemata nel programma. Quello che importa, se sono presenti più DATA, è l'ordine nel quale compaiono, cioè la sequenza dei numeri di linea. Tutte le frasi DATA presenti nel programma contribuiscono alla formazione di una lista di dati nell'ordine nel quale sono nel programma. La prima costante della prima frase DATA è il primo dato nella lista, l'ultima costante dell'ultima frase DATA è l'ultimo dato nella lista. Non appena il programma caricato in memoria viene mandato in esecuzione, il sistema predispone un puntatore interno al primo dato della lista generata dalle frasi DATA. Tale puntatore si sposta ogni volta che viene prelevato un dato dalla lista per mezzo della istruzione READ (lettura interna al programma) che lo trasferisce in una variabile. Esiste una istruzione che permette di riposizionare il puntatore all'inizio della lista dei dati, la RESTORE.
I dati della lista possono essere di qualunque tipo, le stringhe devono essere delimitate dalle virgolette (apici) solo se contengono i seguenti caratteri:

, : spazi lettere digitate insieme a SHIFT
caratteri grafici caratteri di controllo

devi stare attento quando usi l'istruzione READ per attingere alla lista dei dati e usare nomi di variabili che si accordino con il tipo di variabile puntata al momento.

Usare sempre variabili stringa toglie dagli impicci e non fa rischiare errori.
Questa istruzione non può essere usata in modo immediato.

Segue un esempio di uso di DATA:

```
1 REM ES11 PROVA DATA
10 DATA"PROVA DI DATA","PASSO AI NUMERI"
20 DATA1,2,3,4,5,6,7,8,9,0
30 DATA32767,-32768,127E-14
40 DATAA,B,C,D,E,F,G,H,I,J,K,L,M
50 DATAN,O,P,Q,R,S,T,U,V,W,X,Y,Z
60 DATA"[illegible]"
70 DATAFINE
```

Con le frasi riportate si definisce una lista di 43 costanti.
Nota che sono state scritte tra virgolette le prime due stringhe, infatti contengono degli spazi, la penultima che contiene caratteri di controllo. L'ultima costante, la parola FINE, non necessita di virgolette, ma non sarebbe un errore usarle.

DEF FN Tipo: Servizio

Formato: DEF FN nome(var)=espressione

dove:
DEF FN può essere scritto anche senza spazio DEFFN,
nome è il nome che si vuole assegnare alla funzione che si sta definendo e deve rispettare le regole di formazione dei nomi delle variabili numeriche decimali; la funzione viene richiamata da: FNnome;
var è una variabile che serve come argomento falso per definire matematicamente la funzione; al momento dell'uso la funzione viene richiamata ed eseguita con l'argomento (variabile o costante) che gli viene passato, e che diventa l'argomento vero;
espressione è una espressione di calcolo, nella quale possono essere richiamate anche altre funzioni.

E' consentito l'uso di una sola variabile falsa (dummy); al momento del richiamo della funzione si deve porre tra parentesi l'argomento da usare nel calcolo, che può essere una costante o una variabile. La definizione della funzione, cioè l'esecuzione della linea di programma che la definisce deve avvenire una volta, prima che la funzione possa essere richiamata.
Le funzioni definite dall'utente, dopo la definizione nell'ambito del programma che le usa, si comportano allo stesso modo di quelle fornite dal Basic.

Non è consentito definire una funzione in modo immediato, se si prova si ha il messaggio: ?ILLEGAL DIRECT ERROR

Segue un programma esempio nel quale sono definite due funzioni, e, nella definizione della seconda, viene richiamata la prima.

```
1 REM ES12 PROVA DEF FN
10 DEFFNF1(Y)=Y↑2
20 DEFFNF2(Z)=3*FNF1(Z)
30 INPUT"ARGOMENTO F1:  ";A
40 INPUT"ARGOMENTO F2:  ";B
50 PRINTFNF1(A),FNF2(B)
60 STOP
```

DIM Tipo: Trattamento interno dati

Formato: DIM var(ind) [,var(ind)...]

dove: var è un nome di variabile nel formato consentito, seguito tra parentesi dai valori massimi di ognuno degli indici, separati da virgole.

Deve essere usata questa istruzione prima di richiamare nel programma le variabili con indice, qualora almeno uno degli indici superi il valore 10. Se non si procede così si ha la segnalazione: ?OUT OF RANGE. L'istruzione DIM deve essere eseguita una sola volta per ogni variabile con indice; se essa viene ripetuta si ha il messaggio: ?REDIM'D ARRAY.
Gli indici partono dal valore 0 e possono teoricamente arrivare a 32767; il numero delle dimensioni di una variabile con indice può arrivare a 255, ma si devono fare i conti con la disponibilità di memoria. Puoi vedere nel Capitolo 8 quanta memoria viene consumata definendo variabili con indice.

Segue un esempio nel quale viene dimensionata una tabella di stringhe A$, formata da 7 righe di 3 colonne ciascuna, 21 elementi in totale. Inoltre viene dimensionata una lista di numeri interi A% che può contenere 100 elementi, e una tabella a 3 dimensioni di numeri con decimali A, nella quale il primo indice varia da 0 a 4, il secondo da 0 a 5 e il terzo da 0 a 6, si hanno 210 elementi.

```
1 REM ES13 PROVA DIM
10 DIMA$(6,2)
20 DIMA%(99),A(4,5,6)
```

Le variabili con indice si richiamano ponendo tra parentesi dopo il nome, delle costanti o delle variabili, che definiscono quale elemento si vuole.

END Tipo: Controllo

Formato: END

Questa istruzione fa terminare l'elaborazione e fa comparire la parola READY. sul video (non compare BREAK ... come con STOP). Il programma può continuare se si scrive CONT e RETURN, sempre che dopo la END siano presenti altre istruzioni. Non ha senso usarla in modo immediato.

EXP Tipo: Funzione matematica

Formato: EXP (numero)

dove numero può anche essere una variabile o un'espressione numerica.

Essa calcola il valore della costante e (base dei logaritmi naturali, numero irrazionale di cui è memorizzata una approssimazione uguale a 2.71828183) elevato alla potenza fornita da numero. Se numero supera 88.0296919 si ha il messaggio: ?OVERFLOW.

Puoi provare il programma che segue per diversi valori di X.

```
1 REM ES15 PROVA EXP
10 INPUT"ESPONENTE PER E: ";X
20 PRINT"E↑";X;"="EXP(X)
30 STOP
```

FN Tipo: Funzione matematica

Formato: FN nome(numero)

dove numero può anche essere una variabile numerica. La funzione con il suo nome è stata definita da una istruzione DEF FN. Se questo non è avvenuto, si ha il messaggio: ?UNDEF'D FUNCTION.
Nella definizione della funzione, cioè nella espressione di calcolo, può anche non comparire la variabile falsa, usata al momento della definizione. In tale caso il valore calcolato non dipende dal numero posto tra parentesi.
Quando viene richiamata una funzione ad essa viene sostituito il suo valore numerico.

Segue un programma nel quale sono definite due funzioni; nella prima la variabile falsa è coinvolta nel calcolo, nella seconda no. Esse sono poi richiamate e viene stampato il risultato.

```
1 REM ES16 PROVA FN
10 DEF FN A(P)=5*P-7
20 DEF FN B(R)=10-A/3
30 PRINT"FN A(6)=";FN A(6)
35 A=2:GOSUB40
37 A=5:GOSUB40
39 STOP
40 PRINT"FN B(0)=";FN B(0)
50 PRINT"FN B(1)=";FN B(1)
55 RETURN
```

RISULTATI PROGRAMMA PRECEDENTE

```
FN A(6)= 23
FN B(0)= 9.33333333
FN B(1)= 9.33333333
FN B(0)= 8.33333333
FN B(1)= 8.33333333


BREAK IN 39
```

Come puoi vedere il valore fornito dalla funzione FN B non dipende dal numero passato tra parentesi, ma dal valore attuale della variabile A coinvolta nel calcolo.

FOR...TO... [STEP...] Tipo: Controllo

Formato: FOR var=lim1 TO lim2 [STEP incr]

dove:

var è il nome di una variabile numerica decimale (floating point), che viene usata come contatore (variabile di controllo),

lim1 e lim2 sono il valore iniziale e il valore finale del contatore var,

incr, che esiste solo se è presente la parola chiave STEP, rappresenta l'incremento da usare per il contatore var, per passare dal valore iniziale al valore finale. Se STEP manca l'incremento è assunto implicitamente uguale a 1. STEP può anche essere seguito da incr negativo; in tale caso lim 1 deve essere maggiore di lim 2.

Questa istruzione serve per INIZIARE L'ESECUZIONE di un CICLO e non può essere usata da sola; il ciclo iniziato da FOR e controllato dalla variabile di controllo (contatore) in base al valore iniziale, al valore finale e all'incremento (STEP), termina con la istruzione NEXT var, dove var è lo stesso nome di variabile che compare nella istruzione FOR. Vediamo cosa si intende con "termina"; si intende che vengono eseguite ciclicamente tutte le istruzioni che sono comprese tra la istruzione FOR e la istruzione NEXT collegata alla FOR. Quello che avviene è precisamente questo:

1) con FOR viene inizializzato il contatore, memorizzato il valore finale e l'incremento;
2) vengono eseguite tutte le istruzioni che seguono fino alla istruzione NEXT;
3) viene sommato al contatore l'incremento memorizzato e viene confrontato il valore raggiunto con il valore finale;
4) se il valore raggiunto non supera il valore finale il programma torna alla istruzione immediatamente seguente la istruzione FOR (incomincia un altro ciclo);
5) se il valore raggiunto supera il valore finale il programma prosegue con la istruzione che viene subito dopo la istruzione NEXT, cioè esce dal ciclo.

Le informazioni necessarie per controllare il ciclo FOR sono memorizzate nell'area di lavoro del sistema chiamata STACK; la variabile che serve come contatore sta nella zona variabili del Basic, insieme alle altre variabili.
Se consideri con attenzione questo modo di operare ti accorgi subito che un ciclo FOR viene percorso almeno una volta, infatti l'analisi sul contatore viene fatta quando si incontra il NEXT. Inoltre non sempre viene raggiunto il valore finale; l'incremento può essere tale da far superare lim2, senza raggiungerlo.

Segue un programma esempio, puoi osservare il programma e i suoi risultati e vedrai confermato quanto si è detto. Osserva che all'uscita dal ciclo la variabile di controllo supera lim2 per incremento positivo e gli è inferiore per incremento negativo.

```
1 REM ES17 PROVA FOR...TO...STEP
5 PRINT"ESECUZIONE CON FOR"
10 FORK=1TO10
20 PRINTK;"  ";:NEXTK:PRINT
23 PRINT"VALORE K ALL'USCITA =";K
25 PRINT:PRINT"ESECUZIONE SENZA FOR"
30 K=1
35 PRINTK;"  ";:K=K+1:IFK<=10THEN35
40 PRINT:PRINT
50 PRINT"CICLO CON INCREMENTO NEGATIVO"
60 FORK=10TO1STEP-1:PRINTK;"  ";:NEXTK
```

```
65 PRINT:PRINT"VALORE K ALL'USCITA =";K
70 PRINT:PRINT"CICLO CON INCREMENTO <1"
75 FORK=1TO3STEP.3:PRINTK;"  ";:NEXTK
77 PRINT:PRINT"VALORE K ALL'USCITA =";K
```

RISULTATI PROGRAMMA PRECEDENTE

```
ESECUZIONE CON FOR
 1    2    3    4    5    6    7    8    9    10
VALORE K ALL'USCITA = 11

ESECUZIONE SENZA FOR
 1    2    3    4    5    6    7    8    9    10

CICLO CON INCREMENTO NEGATIVO
 10   9    8    7    6    5    4    3    2    1
VALORE K ALL'USCITA = 0

CICLO CON INCREMENTO <1
 1   1.3   1.6   1.9   2.2   2.5   2.8
VALORE K ALL'USCITA = 3.1
```

Segue lo schema a blocchi di un ciclo FOR.

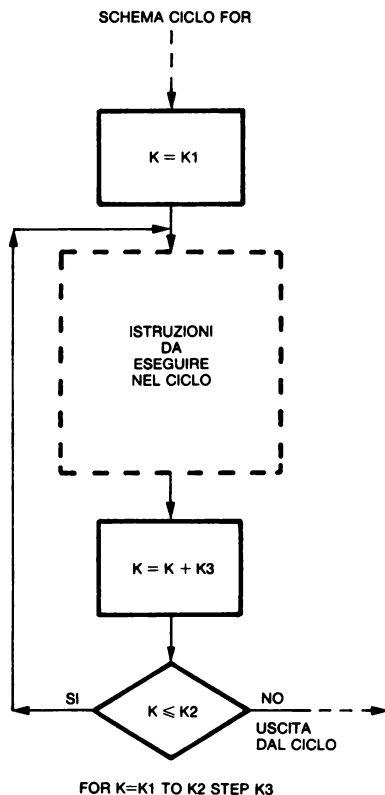
SCHEMA CICLO FOR

K = K1

ISTRUZIONI
DA
ESEGUIRE
NEL CICLO

K = K + K3

SI
K ≤ K2
NO
USCITA
DAL CICLO

FOR K=K1 TO K2 STEP K3

I cicli FOR devono essere usati bene; non si deve uscire da un FOR senza averlo chiuso ponendo la variabile di controllo del ciclo al valore finale e facendo "scattare" il NEXT.
Considera l'esempio che segue:

```
1 REM ES17BIS PROVA FOR
10 DATA13,90,67,456,-987,1234,675
11 DATA1,95,-567,-980,567,23456,689,32
15 FORK=1TO15:READX:PRINTX;:NEXT:PRINT
20 INPUT"NUMERO: ";N
25 RESTORE:FORK=1TO15
30 READX
35 IFX=NTHENPRINT"TROVATO UGUALE":K=15:NEXT:STOP
40 NEXT:PRINT"NON TROVATO UGUALE":STOP
```

in esso si esce dal FOR per una condizione intermedia, ma si opera correttamente. Infatti sia che si esca per esaurimento del ciclo, che per valore trovato uguale, si fa scattare il NEXT. La presenza di due NEXT per un solo FOR non dà fastidio, infatti sono usati in alternativa.
I cicli FOR possono essere concatenati, cioè posti uno interno all'altro, fino ad un massimo di 9.

FRE Tipo: Funzione lettura memoria

Formato: FRE (var)

dove: var può essere un numero o una variabile di qualunque tipo, che non influisce sul risultato della funzione, però deve essere presente.

La funzione fornisce il numero di byte di memoria RAM liberi. Se tale numero è negativo devi sommargli 65536 per ottenere un risultato corretto.

Segue un programma che fornisce il numero dei byte liberi all'inizio, assegna 101 variabili stringa, e poi fornisce di nuovo il numero dei byte liberi. La differenza tra i due risultati dà il numero dei byte usati per le variabili del programma. Nota il modo di calcolare la memoria libera alla linea 100 (sottoprogramma): FRE(0)<0 è una relazione logica che ha valore -1 quando è vera, per cui se FRE(0) risulta negativo si ha che (FRE(0)<0)=—1; in conseguenza 65536 viene sommato al primo FRE(0) e il risultato risulta positivo.

```
1 REM ES18 PROVA FRE
10 GOSUB100:PRINT"BYTE LIBERI INIZIO:";Z
20 DIMA$(100):Z$="PROVA FRE"
30 FORK=0TO100:A$(K)=Z$:NEXTK
40 GOSUB100
50 PRINT"BYTE LIBERI DOPO ASS. VAR.:";Z
60 STOP
100 Z=FRE(0)-(FRE(0)<0)*65536:RETURN
```

RISULTATI PROGRAMMA PRECEDENTE

```
BYTE LIBERI INIZIO: 38714
BYTE LIBERI DOPO ASS. VAR.: 38390


BREAK IN 60
```

Se provi ad eseguire in immediato FRE(0)+ 65536 subito dopo aver acceso il calcolatore trovi il valore 38909.

GET Tipo: Operazione I/O dati

Formato: GET lista-var

dove lista-var è una lista di nomi di variabili separate da virgola, al limite una sola variabile.

Questa istruzione legge un carattere per volta dal buffer della tastiera. Il buffer può contenere 10 caratteri, se si pigiano più di 10 tasti, gli ultimi caratteri vanno persi. Quando si legge con GET un carattere, si crea un posto nel buffer, cioè il carattere letto viene eliminato.
Segue un programma esempio:

```
1 REM ES19 PROVA GET
10 PRINT"SCRIVI UNA PAROLA E"
11 PRINT"TERMINA CON ASTERISCO E RETURN"
15 FORK=1TO3000:NEXTK
20 GETA$:PRINTA$;
30 IFA$="*"THEN50
40 GOTO20
50 PRINT"FINITO PRIMA PROVA"
60 PRINT"ASPETTO UN TASTO PER PROSEGUIRE"
70 GETA$
80 PRINT"PROSEGUO"
90 PRINT"PREMI S PER CONTINUARE, N PER STOP"
93 GETA$:IFA$=""THEN93
95 IFA$<>"S"ANDA$<>"N"THEN93
97 IFA$="S"THEN10
99 STOP
```

La prima prova consiste in questo (linee da 10 a 50):
.. in 10 e 11 viene chiesto di scrivere una parola e di terminare con asterisco; tale parola deve essere corta, cioè entrare nel buffer, in tutto meno di 10 caratteri; vedrai che mentre premi i tasti i caratteri non appaiono;
.. in 15 viene creato un ciclo di attesa, è proprio durante il ciclo nel quale il calcolatore percorre un FOR 3000 volte che tu scrivi e riempi il buffer;
.. in 20, terminato il ciclo, viene prelevato un carattere con GET dal buffer e viene stampato;
.. in 30 e 40 se il carattere stampato è asterisco il programma prosegue da 50, altrimenti torna a 20;

.. da 60 a 80 il programma prosegue solo se si preme un tasto, la istruzione GET A$ della linea 70 lo blocca; ecco un modo per creare una pausa lunga a piacere;
.. a 90 viene chiesto di premere uno tra due tasti, S o N;
.. a 93 GET A$ legge un tasto e prosegue solo quando il tasto viene premuto, infatti se non è premuto alcun tasto A$ è la stringa nulla;
.. a 95 viene controllato il tasto premuto e se non è uno dei due richiesti il programma torna a 93;
.. a 97 se A$ è uguale a S il programma ricomincia, altrimenti si ferma.

Già da questo breve programma hai visto diversi utilizzi di GET. Nel Capitolo 3 vedrai altre applicazioni.
Tieni presente che è meglio far seguire a GET una variabile di tipo stringa, infatti se usi il nome di una variabile numerica e poi premi un tasto non numerico, il sistema segnala un errore.

Questa istruzione non può essere usata in modo immediato.

GET# Tipo: Operazione I/O dati

Formato: GET# lfn, lista-var

dove:
lfn è il numero logico di un file che deve essere stato preventivamente aperto con OPEN;
lista-var è una lista di nomi di variabili separate da virgola, al limite una sola variabile.

Con questa istruzione viene letto un carattere per volta da una periferica diversa dalla tastiera; se non viene ricevuto un carattere e la variabile è di tipo stringa essa diventa una stringa nulla, mentre se è di tipo numerico essa riceve un valore 0.

Segue un programma esempio, nel quale all'inizio (linee da 10 a 30) viene pulito il video e vengono scritte le seguenti 3 righe:

APERTURA VIDEO COME FILE # 1
STO PROVANDO A LEGGERE DAL VIDEO
*

```
1 REM ES20 PROVA GET#
10 PRINT"□ APERTURA VIDEO COME FILE #1"
20 PRINT"STO PROVANDO A LEGGERE DAL VIDEO"
25 PRINT"*"
27 OPEN1,3:REM APRE DEVICE 3 COME FILE 1
```

```
30 PRINT"[HOME]";:Z$=""
40 GET#1,A$:Z$=Z$+A$
45 IFA$="*"THEN60
50 GOTO40
60 PRINT"[HOME][DOWN][DOWN][DOWN][DOWN][DOWN][DOWN][DOWN][DOWN][DOWN]";Z$
65 CLOSE1:STOP
```

poi, alla linea 27 viene aperto il video che è la periferica numero 3, come file logico numero 1. Alla linea 30 con PRINT del carattere HOME si porta il cursore nell'angolo alto a sinistra senza cancellare il video, e si annulla la stringa Z$. Dalla linea 40 alla 50 vengono letti i caratteri delle prime righe del video, uno dopo l'altro con GET# 1, e sommati in Z$. La lettura termina dopo aver incontrato l'asterisco. Alla linea 60 viene portato il cursore in giù e viene stampata la stringa Z$ che contiene tutti i caratteri letti con GET# 1.

Il comando GET# è importante perchè legge tutti i caratteri, anche quelli di controllo, come le virgole separatrici dei campi o il carattere corrispondente al RETURN, CHR$(13).

Questo comando non può essere usato in modo immediato.

GOSUB Tipo: Controllo

Formato: GOSUB numero-linea

dove numero-linea è il numero di una linea, dove inizia un sottoprogramma.

Questa istruzione trasferisce il controllo del programma alla linea indicata (che rappresenta l'inizio di un sottoprogramma) e ricorda, cioè memorizza, il numero della linea in cui è il GOSUB. Il numero di linea che è stato memorizzato viene riesumato, quando nel sottoprogramma si incontra l'istruzione RETURN (che non ha relazione con il tasto RETURN della tastiera) e serve per ritornare all'istruzione subito dopo la GOSUB, riprendendo la sequenza interrotta.

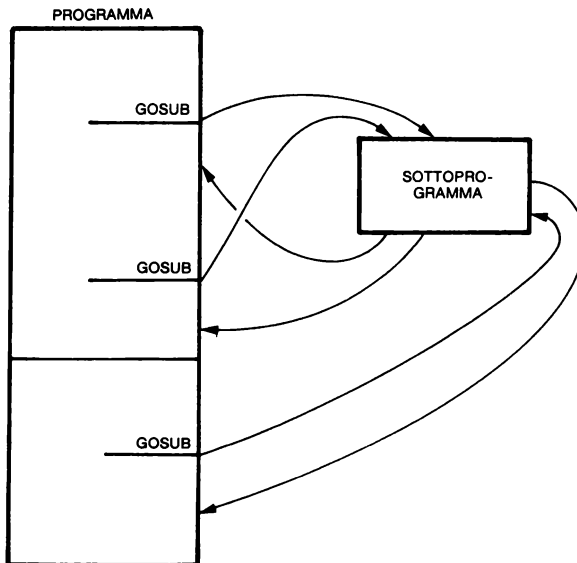
Per sottoprogramma si intende un pezzo di programma che esegue una ben definita sequenza di operazioni e termina con una istruzione RETURN. La programmazione che fa uso di sottoprogrammi risulta molto ordinata e di facile comprensione; infatti ogni sottoprogramma, in genere poche istruzioni, ha una funzione e fornisce risultati ben definiti. Richiamando un sottoprogramma già collaudato, dove serve, si è sicuri di non fare errori, e, inoltre si risparmia molta memoria.
La differenza che esiste tra le funzioni e i sottoprogrammi è che una funzione consiste in una sola formula, mentre un sottoprogramma è formato in generale da più istruzioni.

Segue uno schema a blocchi che esemplifica la chiamata di un sottoprogramma e il ritorno da un sottoprogramma.

SCHEMA GOSUB

PROGRAMMA

GOSUB

SOTTOPRO-
GRAMMA

GOSUB

GOSUB

Segue un semplice esempio di uso di sottoprogrammi:

```
1 REM ES21 PROVA GOSUB
10 PRINT"PROSPETTO ALUNNI CLASSE III A"
20 GOSUB100
25 PRINT"ISCRITTI NUMERO 25"
30 GOSUB110:GOSUB100
35 PRINT"PROMOSSI NUMERO 25"
40 C$="\":GOSUB120
45 C$="/":GOSUB120
99 STOP
100 FORK=1TO40:PRINT"-";:NEXTK:RETURN
110 FORK=1TO40:PRINT"*";:NEXTK:RETURN
120 FORK=1TO40:PRINTC$;:NEXTK:RETURN
```

Alla linea 100 e alla linea 110 sono presenti due sottoprogrammi che tracciano rispettivamente una linea di trattini e una linea di asterischi. Essi sono richiamati in

20 e 30. Alla linea 120 è presente un sottoprogramma che riceve dal programma in C$ il carattere da usare per tracciare la linea.

La memorizzazione del numero di linea, e anche della posizione dell'istruzione nella linea per le istruzioni multiple, ha luogo in una speciale area di lavoro chiamata STACK, di 256 byte. Il meccanismo di utilizzo dell'area STACK è il seguente:
.. al momento del GOSUB viene memorizzata l'informazione per il ritorno come ultima nella pila di informazioni già memorizzate;
.. al momento del RETURN viene prelevata l'informazione che si trova sopra tutte le altre, che risulta essere l'ultima depositata (in inglese si dice LAST IN FIRST OUT, l'ultima messa è la prima tolta);
.. questa gestione consente che un sottoprogramma ne chiami un altro, fino ad un massimo di 23 (concatenamento dei sottoprogrammi).

Segue il programma GOSUBCONC, che dimostra quale è il numero massimo di GOSUB concatenati consentiti. Infatti fino a quando non si esegue il RETURN l'area STACK non viene svuotata, a un certo punto manca spazio e si ha il messaggio di errore: OUT OF MEMORY.

```
1 REM GOSUBCONC
10 K=0:GOSUB100:STOP
100 :K=K+1:PRINTK;:GOSUB100
```

GOTO Tipo: Controllo

Formato: GOTO numero-linea oppure
GO TO numero-linea

Questa istruzione fa proseguire il programma dal numero di linea indicato.
Se il numero di linea non è presente nel programma si ha la segnalazione: ?UNDEF'D STATEMENT ERROR IN .. .
Scrivendo:

120 GOTO 120

si crea un ciclo senza fine che può essere interrotto premendo il tasto RUN/STOP.

IF...THEN... Tipo: Controllo

Formato: IF espressione THEN numero-linea
IF espressione GOTO numero-linea
IF espressione THEN istruzione

dove:

espressione è una qualunque espressione valida nel linguaggio che può essere VERA o FALSA;

la parte di istruzione che viene dopo THEN o GOTO viene eseguita se il risultato di espressione è VERO;

se il risultato di espressione è FALSO, il programma prosegue dal numero-linea immediatamente successivo a quello della linea che contiene l'istruzione IF.

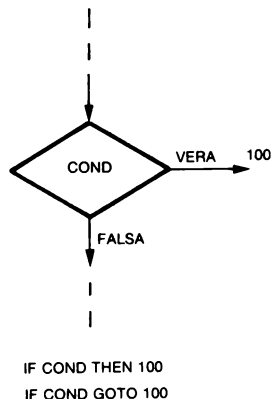
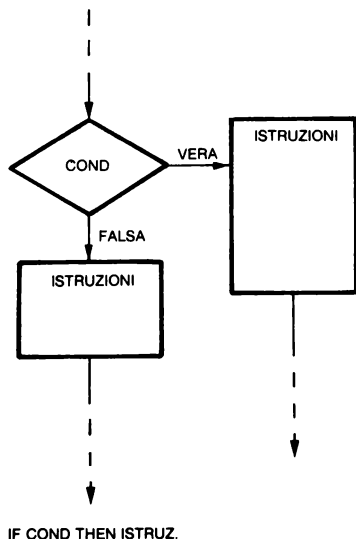
Se si usa il formato con GOTO, la condizione VERA provoca un salto in un determinato punto del programma; siamo in presenza di un salto sotto condizione. La normale istruzione GOTO provoca invece un salto incondizionato.
Il formato THEN numero-linea è sostanzialmente identico al GOTO numero-linea.
Quello che presenta maggiore interesse, in certi casi naturalmente, è il formato THEN istruzione. Infatti dopo il THEN possono essere presenti più istruzioni separate dal carattere separatore due punti. Devi fare attenzione al fatto che se la serie di istruzioni non termina con un GOTO, dopo viene eseguita la stessa sequenza del caso di espressione FALSA.
Naturalmente in questo caso nelle istruzioni dopo THEN può comparire un'altra istruzione IF e bisogna considerare con attenzione il concatenamento delle condizioni VERO e FALSO.

Segue un programma esempio nel quale sono riportati i tre formati dell'istruzione IF.

```
1 REM ES23 PROVA IF
10 INPUT"SCRIVI UN NUMERO: ";N
20 IFN<0THEN40
30 PRINT"NUMERO POSITIVO":GOTO50
40 PRINT"NUMERO NEGATIVO"
50 IFABS(N)>=500GOTO70
60 PRINT"NUMERO MINORE 500":GOTO80
70 PRINT"NUMERO MAGGIORE/UGUALE 500"
80 INPUT"SCRIVI UNA PAROLA: ";A$
90 PRINTA$
95 IFLEN(A$)>10THENPRINT"LUNGH. A$ >10":STOP
97 PRINT"LUNGH. A$ <= 10"
100 STOP
```

Seguono gli schemi a blocchi dei diversi tipi di IF.

IF COND THEN ISTRUZ.

IF COND THEN 100
IF COND GOTO 100

INPUT Tipo: Operazione I/O dati

Formato: INPUT ["messaggio";] lista-var

dove:

messaggio può essere presente e in questo caso deve essere una stringa delimitata dalle virgolette e seguita dal punto e virgola (;); non è ammessa una variabile stringa;

lista-var è una lista di variabili separate da virgola, al limite una variabile sola.

Essa serve per leggere dati dalla tastiera e assegnarli alle variabili.
Rispondendo solo con il tasto RETURN alla richiesta di dato, la variabile coinvolta mantiene il suo precedente contenuto. Per scrivere un dato devi digitarlo e poi premere il tasto RETURN.
L'istruzione provoca la comparsa di un punto interrogativo sul video, per chiedere dati. Devi rispondere digitando i dati e separandoli con virgole se sono più di uno; il RETURN finale chiude il colloquio. Se rispondi con meno dati di quelli richiesti, compaiono due punti interrogativi a significare che non hai soddisfatto la richiesta in modo completo. Se usi come separatore i due punti, viene accettato un dato fino ai due punti e il resto viene ignorato, sia per dati numerici che per stringhe. Se invece usi il punto e virgola viene segnalato errore:?REDO FROM START per i dati numerici e viene richiesto il dato di nuovo; per le stringhe invece il punto e virgola è

accettato come carattere della stringa. Se vuoi comprendere nelle stringhe anche caratteri separatori altrimenti non accettati devi aprire le virgolette prima di iniziare la stringa e chiuderle subito dopo.
Se rispondi con dati stringa a richieste numeriche (accetta solo: segno, cifre e un punto decimale) hai il messaggio: ?REDO FROM START e ti viene chiesto di nuovo il dato.
La lunghezza del messaggio, se presente, non può superare 39 caratteri, infatti il punto interrogativo che compare a chiedere dati non deve passare a nuova linea, altrimenti esso viene incorporato nella risposta. Comunque tra messaggio e risposta non possono essere superate due linee del video, compreso il RETURN finale. Nel caso tu voglia avere un messaggio esplicativo più lungo, puoi scriverlo con PRINT e poi usare INPUT solo per il dato. Se rispondi con più dati di quelli richiesti, compare il messaggio:?EXTRA IGNORED e il programma prosegue; i dati in più non sono stati trasferiti in alcuna variabile e vanno persi.
L'istruzione INPUT non può essere interrotta, cioè non sente il tasto RUN/STOP. Se desideri interrompere una richiesta di dati e fermare il programma, devi premere RUN/STOP e contemporaneamente RESTORE.
Questa istruzione non può essere usata in modo immediato.

Segue un programma con il quale puoi verificare quanto si è detto.

```
1 REM ES24 PROVA INPUT
10 INPUTA:PRINTA
20 INPUTA%:PRINTA%
30 INPUTA$:PRINTA$
40 INPUTA,B,C
50 PRINTA,B,C
60 INPUT"SCRIVI UN NUMERO: ";N
70 PRINTN
80 INPUT"SCRIVI UNA FRASE: ";A$
90 PRINTA$
99 STOP
```

Nel capitolo 3 torniamo diffusamente sull'argomento dell'ingresso e dell'uscita dei dati, colloquio video/tastiera.

INPUT# Tipo: Operazione I/O dati

Formato: INPUT# lfn, lista-var

dove:
lfn è il numero logico di un file che deve essere stato aperto su una periferica con il comando OPEN;
lista-var è una lista di variabili separate da virgole, al limite una variabile sola.

L'istruzione preleva dati dal file considerando terminata una variabile quando incontra un separatore che può essere:
.. il carattere generato dal tasto RETURN, cioè CHR$(13);
.. la virgola;
.. il punto e virgola;
.. i due punti;
comunque una variabile non può essere più lunga di 80 caratteri, compreso il carattere separatore.
Ovviamente un numero non ha mai più di 80 cifre, mentre una stringa può avere più di 80 caratteri. Se si legge una stringa troppo lunga si ha il messaggio: ?STRING TO LONG, mentre se si legge in una variabile numerica una stringa alfanumerica, si ha il messaggio:?BAD DATA.
Una stringa troppo lunga, o anche un numero troppo lungo si può trovare in un file, se questo è stato scritto in modo errato; vai a vedere l'istruzione PRINT# .

Questa istruzione non può essere usata in modo immediato.

Segue un programma esempio, che risulta dalla modifica di quello riportato nella spiegazione dell'istruzione GET# . Qui si apre il video come file e si va a leggere con l'istruzione INPUT# .

```
1 REM ES25 PROVA INPUT#
10 PRINT"APERTURA VIDEO COME FILE #1"
20 PRINT"STO PROVANDO A LEGGERE DAL VIDEO"
25 PRINT"*"
27 OPEN1,3:REM APRE DEVICE 3 COME FILE 1
30 PRINT"";:Z$=""
40 INPUT#1,A$
45 INPUT#1,B$
50 INPUT#1,C$
60 PRINT""
63 PRINTA$,B$,C$
65 CLOSE1:STOP
```

I comandi relativi ai file su cassetta e su floppy disk sono trattati diffusamente nel secondo volume di questa serie di manuali per il COMMODORE 64.

INT Tipo: Funzione matematica

Formato: INT (numero)

dove numero può essere un numero, una variabile numerica o una espressione numerica.

La funzione fornisce la parte intera dei numeri positivi, troncando i decimali, mentre per i numeri negativi con decimali viene fornito il numero negativo immediatamente inferiore (INT(-25.456)=-26).

Segue un programma esempio dove si calcola il resto di una divisione tra due numeri.

```
1 REM ES26 PROVA INT
10 INPUT"PRIMO NUMERO:    ";N1
20 INPUT"SECONDO NUMERO: ";N2
30 N3=N1/N2
40 R=N3-INT(N3)
50 PRINT"PARTE INTERA N1/N2: ";INT(N3)
55 PRINT"PARTE DECIMALE N1/N2: ";N3-INT(N3)
60 PRINT"RESTO DIV. SENZA DEC.: ";N1-N2*INT(N3)
70 PRINT"N1/N2: ";N3
80 STOP
```

LEFT$ Tipo: Funzione stringa

Formato: LEFT$ (stringa, intero)

dove:
stringa è una stringa o una variabile stringa (anche la somma di più stringhe e/o variabili stringa);
intero è un numero intero o una variabile numerica con valore compreso tra 0 e 255.

La funzione fornisce i primi "intero" caratteri della stringa se essa è più lunga, tutta la stringa se essa è più corta.

Provando il programma esempio che segue, puoi introdurre diverse risposte per A$ e studiare il comportamento della funzione.

```
1 REM ES27 PROVA LEFT$
5 S$="                    "
10 INPUT"SCRIVI UNA FRASE: ";A$
20 B$=LEFT$(A$+S$,20)
25 C$=LEFT$(A$,3)
30 PRINTB$;"*"
35 PRINTC$
40 STOP
```

Alla linea 20, sommando ad A$ la stringa S$ di 20 spazi, si è sicuri che la B$ risulta di 20 caratteri. Infatti se si opera con la funzione su una stringa più corta della lunghezza richiesta, la stringa risultato non ha la lunghezza richiesta. Alla linea 30 si stampa un asterisco subito dopo B$ per metterne in evidenza la lunghezza anche se contiene spazi.

LEN Tipo: Funzione lettura memoria

Formato: LEN (stringa)

dove stringa può essere una stringa o una variabile stringa.

Essa fornisce il numero dei caratteri che compongono la stringa.

Segue un semplicissimo programma per provare la funzione.

```
1 REM ES28 PROVA LEN
10 INPUT"STRINGA: ";A$
20 PRINTLEN(A$)
30 STOP
```

LET Tipo: Trattamento interno dati

Formato: [LET] var=espressione

dove: LET può essere omessa;
 var è il nome di una variabile consentita dal Basic;
 espresssione è una espressione compatibile con il tipo della variabile che compare a sinistra dell'uguale.

Questa istruzione serve per assegnare dati alle variabili. Se si tratta di variabili numeriche, espressione può essere un semplice numero o una complicata espressione di calcolo. Se si tratta di variabili stringa, espressione può essere una stringa delimitata da virgolette, una variabile stringa o una somma di stringhe e/o variabili stringa.

Segue un programma nel quale si fa vedere che usando e non usando la parola chiave LET si ottengono gli stessi risultati.

```
1 REM ES29 PROVA LET
10 INPUT"PRIMO NUMERO    :";N1
20 INPUT"SECONDO NUMERO :";N2
```

```
30 LETM=N1+N2-(N1/(2+INT(N2)))
40 P=N1+N2-(N1/(2+INT(N2)))
45 PRINT"M=";M:PRINT"P=";P
50 STOP
```

LIST Tipo: Servizio

Formato: LIST [[prima.linea] [—ultima.linea]]

Serve per listare sul video il programma presente in memoria.

LIST senza numeri di linea, lista tutto il programma; si ha lo scrolling del video durante la lista, per rallentare il movimento si può tenere premuto il tasto CTRL

LIST numero.linea lista una sola linea

LIST numero.linea- lista a partire da numero.linea

LIST -numero.linea lista fino a numero.linea, partendo dall'inizio

LIST numero1-numero2 lista dalla linea numero1 alla linea numero2.

Se vuoi fermare la lista devi premere il tasto RUN/STOP.

Usando le istruzioni OPEN e CMD si può dirigere la lista verso un'altra periferica, in particolare verso la stampante.

Questa istruzione può essere usata anche all'interno di un programma, solo che, alla fine della lista richiesta, non restituisce più il controllo al programma. Se premi CONT, ottieni ancora la lista del programma.

LOAD Tipo: Operazione I/O programmi

Formato: LOAD [fn] [,dn] [,ind]

dove:
fn (stringa tra virgolette) è il nome del file da caricare in memoria; può anche essere il nome di una variabile stringa che contiene il nome del file;
dn è il numero logico della periferica da cui prelevare il programma;
ind è un numero che fornisce informazioni sull'indirizzo di memoria da cui iniziare il caricamento del file.

Serve per caricare programmi in memoria prelevandoli da nastro o da disco. Il nastro è di norma individuato da dn=1, il disco da dn=8.
Di solito i programmi Basic vengono caricati in memoria a partire dall'indirizzo 2049, ponendo 1 per ind, si può caricarli partendo dall'indirizzo dal quale partivano prima di essere memorizzati con il comando SAVE. Il puntatore all'inizio del programma si trova nella pagina 0 della RAM nei byte 43 e 44 e può essere modificato.
Con i comandi LOAD, SAVE e VERIFY, che riguardano operazioni con i programmi, non si deve usare prima l'istruzione OPEN.
Se si usa un nome di programma inesistente, si ha il messaggio: ?FILE NOT FOUND.
Elenchiamo i possibili formati di LOAD.

LOAD senza parametri, legge il primo programma che trova sul nastro.

LOAD N$ oppure LOAD "nome"
legge da nastro il programma con il nome fornito.

LOAD "",1,1 oppure LOAD n$,1,1 oppure LOAD "nome",1,1
legge da nastro il primo programma che trova o il programma avente il nome fornito, ma lo carica a partire dall'indirizzo dal quale iniziava quando è stato memorizzato (non opera una rilocazione del programma).

LOAD "*",8 legge da disco il primo programma, cioè quello che compare per primo nella DIRECTORY del disco.

LOAD "nome",8 oppure LOAD n$,8
legge da disco il programma con il nome fornito.

LOAD "nome",8,1 oppure LOAD n$,8,1 legge da disco il programma con il nome fornito senza rilocarlo.

Quando si dà il comando LOAD si ha un colloquio tramite video, e precisamente:

.. per il nastro:

PRESS PLAY ON TAPE
FOUND nome...
LOADING
READY.

se il programma non è il primo, si ha una ripetizione di FOUND... fino ad arrivare al nome cercato; dopo il primo messaggio si deve avviare il registratore premendo il tasto PLAY;

.. per il disco:
SEARCHIN FOR nome
LOADING
READY.

L'istruzione LOAD può essere data in modo immediato; in questo caso il sistema prima di caricare il programma chiude tutti i file aperti e azzera tutte le variabili. Quando si usa LOAD dall'interno di un programma, il nuovo programma caricato, parte automaticamente; in questo modo si possono concatenare tra loro più programmi e non vengono azzerate le variabili, ma si deve fare attenzione alla lunghezza dei programmi per non perdere variabili a causa di sovrapposizioni. Vedi il Capitolo 4 in merito.

LOG Tipo: Funzione matematica

Formato: LOG (argomento)

dove argomento può essere un numero o una espressione numerica.

Calcola il logaritmo naturale, cioè in base e, dell'argomento.
L'argomento deve essere diverso da zero e positivo, altrimenti si ha il messaggio: ?ILLEGAL QUANTITY.

Segue un esempio; puoi provare a calcolare diversi logaritmi.

```
1 REM ES32 PROVA LOG
5 A$="LOG(";N;") IN BASE 10 ="
10 INPUT"NUMERO: ";N
20 PRINT"LOG(";N;")=";LOG(N)
30 PRINTA$;LOG(N)/LOG(10)
40 PRINT"LOG(2.71828183)="LOG(2.71828183)
50 PRINT"LOG(57/3+78)=";LOG(57/3+78)
60 STOP
```

MID$ Tipo: Funzione stringa

Formato: MID$ (stringa, int-1 [,int-2])

dove:
stringa può essere una stringa delimitata da virgolette, una variabile stringa o una somma di stringhe e/o di variabili stringa;
int-1 è un intero o una variabile intera che indica da quale carattere della stringa

(dà la posizione) iniziare l'estrazione della sottostringa;
int-2, se presente indica quanti caratteri estrarre; se int-2 manca si intende che si arriva alla fine della stringa.

Serve per estrarre parti di stringa da una stringa. Se int-1 manda fuori dalla stringa o int-2 è uguale a zero, la stringa estratta è la stringa nulla. Se int-2 è troppo grande vengono estratti meno caratteri.

Segue un programma esempio:

```
1 REM ES33 PROVA MID$
5 R$="4 CARATTERI CENTRALI: "
10 INPUT"STRINGA: ";A$
20 L=LEN(A$)
25 IFL<4THENSTOP
30 PRINT"PRIMI 3 CARATTERI: ";MID$(A$,1,3)
40 PRINT"ULTIMI 3 CARATTERI: ";MID$(A$,L-2)
50 PRINTR$;MID$(A$,L/2-1,4)
60 STOP
```

Ti viene chiesta una stringa, viene calcolata la lunghezza della stringa e vengono stampati i primi tre caratteri, gli ultimi tre caratteri e i quattro caratteri centrali.

NEW Tipo: Servizio

Formato: NEW

Serve per cancellare il programma Basic presente in memoria e tutte le sue variabili. Si deve usare prima di iniziare la scrittura di un programma. Si può usare anche come istruzione di programma, ma quando viene eseguita il programma che la contiene viene cancellato e quindi non va oltre questa istruzione.

NEXT Tipo: Controllo

Formato: NEXT var [,var]...

dove var sono nomi di variabili inizializzate come contatori in un ciclo FOR.

Serve per delimitare il campo di azione di un ciclo FOR. Quando viene incontrato NEXT, il contatore di ciclo viene incrementato (o decrementato) e confrontato con il limite per stabilire se continuare il ciclo o uscirne.
I cicli FOR possono essere concatenati tra loro fino a 9 volte, cioè 9 cicli uno interno all'altro. La parola chiave NEXT può anche comparire senza il nome della variabi-

le, in questo caso viene riferita all'ultimo ciclo FOR aperto. Si ha invece un messaggio di errore: ?NEXT WITHOUT FOR, se si cita in modo errato la variabile. Inoltre se più cicli FOR concatenati si chiudono insieme, si può scrivere una sola parola chiave NEXT seguita dai nomi delle diverse variabili coinvolte separate da virgola; naturalmente i nomi delle variabili devono essere scritti nel giusto ordine, prima quella del FOR aperto per ultimo e così via.

Segue un programma esempio dove puoi vedere applicate alcune delle regole esposte.

```
1 REM ES35 PROVA NEXT
10 PRINT"TABELLA DI NUMERI▮"
20 FORK=1TO10
23 FORJ=1TO6
24 X=K*J:X$=STR$(X):IFLEN(X$)<3THENX$=" "+X$
26 PRINTX$;"  ";
27 NEXTJ:PRINT
29 NEXTK
30 PRINT"▮RIEMPIE MATRICE A 3 DIMENSIONI▮"
31 DIMZ(3,4,5)
35 FORK=0TO3:FORJ=0TO4:FORL=0TO5
40 Z(K,J,L)=K*J*L
45 NEXTL,J,K
50 INPUT"QUALE ELEMENTO (K,J,L):";K,J,L
55 PRINT"Z(";K;",";J;",";L;")=";Z(K,J,L)
60 STOP
```

Nota alla linea 24 l'accorgimento usato per incolonnare i numeri con diverso numero di cifre. Se vuoi vedere altri elementi della matrice Z(k,j,l), puoi scrivere in immediato GOTO 50.

ON Tipo: Controllo

Formato: ON espressione GOTO/GOSUB lista-numeri-linea

dove:

espressione può essere una qualunque espressione numerica che dia un risultato non negativo da 1 in avanti del quale viene considerata solo la parte intera;

lista-numeri-linea è una lista di numeri di linea presenti nel programma, separati da virgole.

Serve per saltare a linee di programma con GOTO, a sottoprogrammi con GOSUB,

a seconda del valore numerico di espressione. Se il valore è 1, il salto avviene al primo numero di linea della lista, se esso è 2, al secondo, se esso è N, all'ennesimo. Se il valore è zero o supera il numero di numeri di linea presenti, il programma prosegue dalla linea di programma seguente. Un valore negativo dell'espressione provoca il messaggio: ?ILLEGAL QUANTITY.

Questa istruzione sostituisce, se si trova l'algoritmo adatto per scrivere l'espressione, una catena di IF...THEN.

Segue un programma esempio.

```
1 REM ES37 PROVA ON
10 INPUT"]■■SCEGLI: A,B,C,D ";X$
15 IFASC(X$)>=65ANDASC(X$)<=68THEN20
17 GOTO10
20 ONASC(X$)-64GOTO100,200,300,400
100 PRINT"SCELTO A":STOP
200 PRINT"SCELTO B":STOP
300 PRINT"SCELTO C":STOP
400 PRINT"SCELTO D":STOP
```

Segue un diagramma esplicativo dell'istruzione ON.

SCHEMA DIRAMAZIONE ON

K=1 ISTRUZ. N1

K=2 ISTRUZ. N2

K=0
K>4

K

K=3 ISTRUZ. N3

K=4 ISTRUZ. N4

ISTRUZ.
SUCCESSIVA

ON K GOTO N1, N2, N3, N4

CORRISPONDE A IF IN CASCATA

K=1 SI GOTO N 1

NO

K=2 SI GOTO N 2

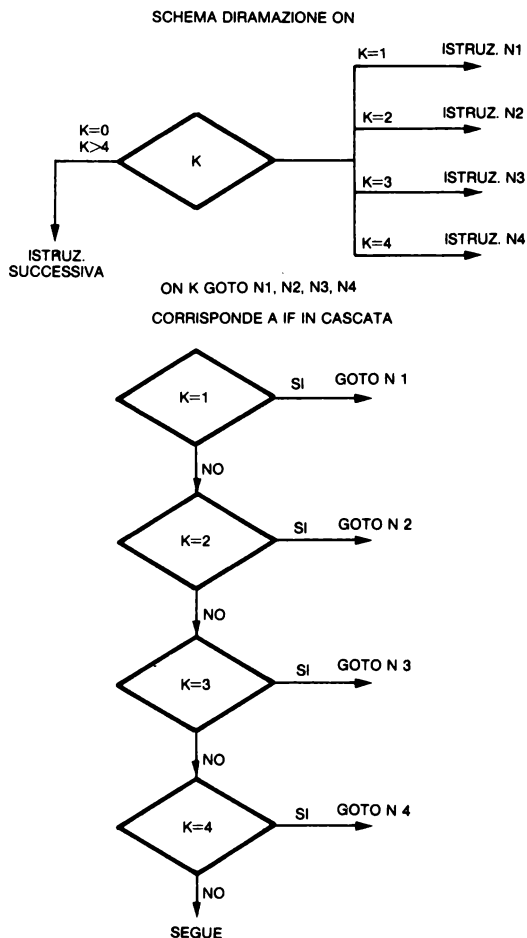
NO

K=3 SI GOTO N 3

NO

K=4 SI GOTO N 4

NO

SEGUE

OPEN Tipo: Operazione I/O dati

Formato: OPEN lfn,dn [,sa] [,"fn [,ft] [,md] "]

dove i parametri hanno il significato spiegato all'inizio di questo paragrafo.

Questa istruzione serve per aprire un file logico, assegnandogli il numero lfn, sulla periferica contraddistinta dal numero dn. Questi primi due parametri devono essere presenti dopo la parola chiave OPEN, se il secondo è omesso, esso viene

assunto uguale a 1. Gli altri possono anche mancare, e, a seconda della periferica, possono avere diversi significati.
Il numero lfn è citato in tutte le operazioni di I/O relative al file. Esso può variare da 1 a 255, ma solitamente si usano numeri al disotto di 127, in quanto gli altri possono avere speciali significati per particolari periferiche.
Il numero dn per le periferiche usuali è riportato nella tabellina posta all'inizio di questo paragrafo.
Il numero sa, se presente, specifica ulteriormente il tipo di operazione per la quale si apre il file.
Il nome del file non è sempre necessario.

Quando viene omesso il dn, esso viene assunto uguale a 1 e quindi la OPEN è verso il registratore a nastro. Per questa periferica i valori di sa sono:
.. 0 per operazioni di lettura (INPUT...)
.. 1 per operazioni di scrittura senza segnalazione di fine nastro
.. 2 per operazioni di scrittura con segnalazione di fine nastro (end-of-tape); questa segnalazione di fine nastro che viene registrata, fa sì che non si possa, per errore, andare a leggere dopo la fine del file, cosa che provoca il messaggio: ?DEVICE NOT PRESENT.
Per operazioni su nastro è bene che nella OPEN sia presente anche il nome del file fn.
Nel caso delle operazioni su disco, dn è uguale a 8 (se è presente una sola unità); sa può variare tra 2 e 14 ed è necessario il nome del file fn. Per informazioni complete sulle operazioni disco, si rimanda all'apposito volume di questa serie.

Per le operazioni sulla stampante si rimanda al Capitolo 5.

Il nome del file, fn, è una stringa di al massimo 16 caratteri; può essere una variabile.

Se non si apre un file e si usano le altre operazioni di I/O si ha il messaggio: ?FILE NOT OPEN. Se si apre un file già aperto si ha il messaggio: ?FILE OPEN. Se si apre un file per leggere ed esso non esiste si ha il messaggio: ?FILE NOT FOUND. Se si apre un file per scrivere ed esso esiste già (periferica disco solo) si ha il messaggio: ?FILE EXISTS.

Riportiamo qui solo pochi esempi di OPEN, rimandando per gli altri ai capitoli o volumi relativi alle periferiche.

OPEN1,3 apre il video come Input o come Output

OPEN2,0 apre la tastiera come Input

Segue il programma VIDEOFILE, nel quale si apre il video come file e si eseguono operazioni di lettura e scrittura.

```
1 REM VIDEOFILE
10 OPEN3,3
15 PRINT#3,"♥";"ABCDEFGHIJKLMNOPQRSTUVWXYZ"
17 A$=""
20 PRINT#3,"S";
23 FORK=1TO26:GET#3,B$
25 A$=A$+B$
30 NEXTK:PRINT#3
35 PRINT#3,"QQQQQQQQQQQQQQQQQQQQQQQQ"A$
40 CLOSE3:STOP
```

PEEK Tipo: Funzione lettura memoria

Formato: PEEK (indirizzo)

dove indirizzo deve essere un numero tra 0 e 65535, cioè un indirizzo di memoria compreso in 64K.

Serve per leggere contenuti dei byte della memoria. Il numero letto varia da 0 a 255. Indirizzi fuori dai limiti provocano il messaggio: ?ILLEGAL QUANTITY.

Segue un programma che legge alcuni puntatori e alcuni byte.

```
1 REM ES40 PROVA PEEK
5 A=PEEK(43)+256*PEEK(44)
10 PRINT"PUNTATORE INIZIO BASIC :";A
15 B=PEEK(45)+256*PEEK(46)
20 PRINT"PUNTATORE INIZIO VARIABILI :";B
30 PRINT"CONTENUTO BYTE 0: ";PEEK(0)
40 PRINT"CONTENUTO BYTE 65535: ";PEEK(65535)
50 STOP
```

POKE Tipo: Trattamento interno dati

Formato: POKE indirizzo, valore

dove: indirizzo è l'indirizzo di un byte;
valore è un numero tra 0 e 255.

Serve per scrivere un numero in un byte. Se non si rispettano i limiti per i due parametri si ha il messaggio: ?ILLEGAL QUANTITY.

E' una istruzione da usare con cautela, può produrre risultati indesiderati. E' molto utile quando si programma in Assembler, ma si può usare anche con succcesso in Basic. Nel programma esempio che segue, si scrivono le 26 lettere dell'alfabeto sulla prima riga del video e si colorano alternativamente in porpora e verde, andando a scrivere i D/CODE nelle posizioni della mappa video e i codici colore nelle corrispondenti posizioni della mappa colori.

```
1 REM ES41 PROVA POKE
10 MC=55296:MV=1024:PRINT"■■■■■■■■■";
20 FORK=1TO26:POKEMV+K-1,K:NEXTK
30 FORK=1TO26STEP2:POKEMC+K-1,4:POKEMC+K,5
35 NEXTK:STOP
```

POS Tipo: funzione lettura memoria

Formato: POS (argomento)

dove argomento è puramente formale e non ha effetto, puoi usare sempre zero.

Fornisce la posizione del cursore sul video, in senso orizzontale; ma considerando una linea video come formata da due righe, 40 + 40 caratteri, dà quindi come risultato un numero compreso tra 0 e 79.
Segue un semplice programma che ne illustra il significato.

```
1 REM ES42 PROVA POS
10 PRINT"■ABCD■■■■EFGH■■■■IJKL■■■■";
20 PRINT"POS";POS(0)
30 INPUT"SCRIVI UNA FRASE: ";A$
40 PRINT"■■■";A$;" POS";POS(0)
50 STOP
```

PRINT Tipo: Operazione I/O dati

Formato: PRINT lista

dove lista può comprendere una e/o più variabili e/o costanti, separate tra loro da caratteri separatori e/o funzioni che influiscono sulla stampa e/o caratteri di controllo specifici.

Questa istruzione è principalmente diretta al video, ma il comando CMD può trasferire il suo effetto a altre periferiche, come abbiamo già visto.

Della PRINT diretta alla stampante dopo un comando CMD parliamo diffusamente nel Capitolo 5.

Elenchiamo qui alcune delle caratteristiche dell'istruzione, rimandando per maggior approfondimenti al Capitolo 3.

Se nella lista compare più di un elemento da stampare, si deve usare come carattere separatore il punto e virgola o la virgola o lo spazio, considerando le regole che seguono:

.. la virgola si può usare per separare due elementi qualsiasi, essa fa spaziare alla prossima zona di stampa; ogni zona di stampa sul video è di 10 caratteri, 4 zone per riga;
.. il punto e virgola si può usare per separare due elementi qualsiasi, esso non provoca spaziature tra gli elementi;
.. lo spazio si può usare solo tra costanti stringa o numeriche, non può essere usato quando uno degli elementi è una variabile numerica, cioè in tutti quei casi in cui l'interpretazione può essere ambigua.

Ogni elemento viene stampato con il suo formato, che per le stringhe è la stringa stessa, mentre per i numeri si ha uno spazio o il segno meno prima del numero e uno spazio dopo. Un numero di una cifra occupa 3 posizioni.
Nella lista di stampa le variabili stringa o le costanti stringa possono comparire una dopo l'altra senza separatori, cioè può essere omesso il punto e virgola o lo spazio.
La stampa comincia sempre dove si trova il cursore sul video. La posizione del cursore può essere modificata usando caratteri di controllo all'interno della lista.
Se la lista di stampa termina senza punteggiatura il cursore si posiziona, dopo la stampa, sulla linea seguente. Se invece la lista di stampa termina con virgola o punto e virgola, essi agiscono secondo le loro regole e quindi può anche succcedere che il cursore resti nella stessa riga dopo aver stampato tutta la lista. La punteggiatura a fine lista inibisce l'invio dei due caratteri ritorno-carrello e alimentazione-linea che sono altrimenti inviati.
Una istruzione PRINT senza lista fa stampare una linea bianca.
Durante la stampa di una lista, se i caratteri sono più di 40, la stampa prosegue sulla linea seguente.

Negli esempi di programmi esaminati fino ad ora abbiamo usato spesso il comando PRINT; per questa ragione non poniamo qui altri esempi e rimandiamo al Capitolo 3.

PRINT# Tipo: Operazione I/O dati

Formato: PRINT# lfn, lista

dove: lfn è il numero logico del file usato nella OPEN;
lista è una lista di variabili.

Questa operazione serve per scrivere su un file associato a una periferica. Per la periferica stampante rimandiamo al Capitolo 5. Per le periferiche nastro e disco l'argomento viene approfondito nel volume ad esse dedicato.
Qui ci limitiamo ad osservare che nella lista delle variabili si deve aver cura di forzare dei separatori tra i dati, in modo che essi vengano registrati sul supporto magnetico come caratteri separatori, e come tali riconosciuti in fase di lettura dati con INPUT# . Infatti i normali separatori punto e virgola e virgola o spazio tra le variabili della lista, agiscono a livello di aggiunta di caratteri spazio ai caratteri dei dati, ma non producono separatori riconoscibili in fase di lettura. Per produrre separatori riconoscibili, tra gli elementi della lista di stampa devono comparire in forma di stringa il RETURN (CHR$(13)), la virgola ("," o CHR$(44)) o il punto e virgola (";" o CHR$(59)).

READ Tipo: Trattamento interno dati

Formato: READ var [,var]....

dove var sono nomi di variabili, separati da virgole se più di uno.

Questa istruzione consente di trasferire ordinatamente i dati forniti dalle istruzioni DATA presenti nel programma in variabili. I nomi delle variabili devono concordare con il tipo dei dati che vengono via via prelevati. In caso di errore si ha il messaggio: ?SYNTAX ERROR AT LINE N, dove N è il numero della linea del DATA coinvolto e non della istruzione READ. Qualora si cerchi di prelevare più dati di quelli immagazzinati, si ha il messaggio: ?OUT OF DATA.

L'uso di READ e DATA consente di attingere rapidamente a costanti trasferendole in variabili, senza dover ricorrere all'istruzione LET, e lavorando solo all'interno del programma. Inoltre il comando RESTORE rende di nuovo disponibili i dati dall'inizio e può essere usato sia quando la lista di dati è esaurita che in qualunque altro momento.

Segue un esempio di uso.

```
1 REM ES45 PROVA READ
10 DIMM$(12):S$="         "
20 FORK=1TO12
30 READM$(K):M$(K)=LEFT$(M$(K)+S$,9)
40 NEXTK
50 FORK=1TO12:PRINTM$(K);" ";:NEXTK
```

```
60 PRINT:STOP
100 DATAGENNAIO,FEBBRAIO,MARZO,APRILE
101 DATAMAGGIO, GIUGNO,LUGLIO,AGOSTO
102 DATASETTEMBRE,OTTOBRE,NOVEMBRE,DICEMBRE
```

Come vedi abbiamo creato un magazzino di dati con i nomi dei mesi, usando 3 DATA, poi li abbiamo trasferiti nella variabile con indice M$, usando l'indice partendo da uno invece che da zero per avere la usuale corrispondenza, e abbiamo aggiunto degli spazi a destra ai nomi lunghi meno di nove caratteri.

REM Tipo: Servizio

Formato: REM annotazioni

Serve per introdurre commenti nei programmi e facilitarne la lettura. Dopo la parola chiave REM si può scrivere quello che si vuole usando qualunque carattere e anche le parole chiave del Basic.

Segue un programma esempio:

```
1 REM ES46 PROVA REM
10 REM PROGRAMMA PROVA
20 PRINT"TITOLO PROGRAMMA":PRINT
30 INPUT"NOME: ";N$:REM CHIEDE NOME
40 REM STAMPA:PRINT"NOME: ";N$
50 PRINT"STOP"
60 REM ******************************
70 REM     PROGRAMMA CALCOLO
80 REM ******************************
90 REM
95 ...........
```

Come vedi dalla linea 60 alla 90 c'è un tentativo di impaginazione estetica del programma. L'istruzione PRINT della linea 40 non verrà MAI eseguita, infatti al primo REM incontrato in una linea di programma essa viene saltata tutta nell'esecuzione. La linea 30 fa invece un uso corretto della REM ponendola dopo l'istruzione da eseguire.
Anche nelle frasi REM non si devono superare 2 righe video, massimo 80 caratteri.
Si può, ma non ha molto senso, usarla in modo immediato.

RESTORE Tipo: Trattamento interno dati

Formato: RESTORE

Questa istruzione serve per riportare il puntatore interno all'inizio della lista dei dati che sono incorporati nel programma per effetto delle istruzioni DATA. Riportare il puntatore all'inizio significa rendere nuovamente disponibili dall'inizio tutti i dati e quindi poter usare di nuovo l'istruzione READ. Nell'esempio che segue, con la DATA sono disponibili 5 costanti stringa; esse vengono stampate sul video, senza la RESTORE della linea 50 non si potrebbe eseguire GOTO10, si avrebbe infatti il messaggio: ?OUT OF DATA.
Vai a vedere nel Capitolo 8 il programma RESTORE, nel quale ti mostriamo come ottenere prestazioni superiori a quelle del comando RESTORE disponibile in questa implementazione del Basic.

```
1 REM ES47 PROVA RESTORE
2 M$="████████████████████ANCORA (S/N):  "
5 DIMD$(5):OPEN2,0
6 REM APERTURA TASTIERA PER INPUT
7 REM SI EVITA ? IN INPUT
10 PRINT"██    █ELENCO ARGOMENTI██"
20 FORK=1TO5:READA$:PRINT"    "A$" ";
25 INPUT#2,D$(K):PRINT
30 NEXTK
40 GOSUB300
45 GOSUB100:IFR$="N"THENSTOP
50 RESTORE:GOTO10
90 DATAPRIMO, SECONDO, TERZO
91 DATAQUARTO,QUINTO
100 PRINTM$;:INPUTR$
105 IFR$<>"S"ANDR$<>"N"THEN100
110 RETURN
300 PRINT"█ELABORAZIONE ARGOMENTI"
301 RETURN
```

RETURN Tipo: Controllo

Formato: RETURN

Questa istruzione (da non confondersi con il tasto RETURN) provoca l'uscita da un sottoprogramma e il ritorno al programma che ha eseguito il GOSUB. Essa va a

prelevare dall'area STACK l'ultimo indirizzo, quello affiorante, e lo usa per tornare alla linea di programma da dove proseguire. Se viene incontrato un RETURN in un programma si ha un messaggio di errore: ?RETURN WITHOUT GOSUB. Segue un semplice programma esempio:

```
1 REM ES48 PROVA RETURN
10 PRINT"VADO A SUBROUTINE":GOSUB100
20 PRINT"SONO TORNATO"
30 REM ERRATO NON STA IN UNA SUBROUTINE
31 RETURN
40 PRINT"QUI NON ARRIVO"
100 PRINT"SONO IN UNA SUBROUTINE":RETURN
```

RIGHT$ Tipo: Funzione stringa

Formato: RIGHT$ (stringa, numero)

dove:

stringa è una stringa, o una variabile stringa, o la somma di stringhe e/o variabili stringa;

numero è una espressione numerica compresa tra 0 e 255; se non intera i decimali vengono ignorati.

Fornisce il numero di caratteri richiesto, prelevandolo a destra della stringa. Se sono chiesti zero caratteri fornisce la stringa nulla. Se sono chiesti più caratteri di quelli che formano la stringa, essa viene fornita per intero.

Segue un semplice esempio.

```
1 REM ES49 PROVA RIGHT$
5 M$="STRINGA SENZA LA PRIMA META': "
10 INPUT"STRINGA: ";A$
20 PRINT"STRINGA INTERA: ";A$
30 PRINT"ULTIMI 5 CARATTERI: ";RIGHT$(A$,5)
40 PRINTM$;RIGHT$(A$,LEN(A$)/2)
50 STOP
```

RND Tipo: Funzione matematica

Formato: RND (argomento)

dove argomento è un numero.

Fornisce un numero pseudo-random prelevandolo dalla sequenza generata dal calcolatore mediante un algoritmo pseudo-casuale.
La sequenza dei numeri viene generata partendo da un numero iniziale che viene chiamato seme (seed). Il seme viene inizializzato al momento dell'accensione del calcolatore.
L'argomento agisce in questo modo:
.. se esso è negativo, predispone in base al suo valore il punto di partenza della sequenza dei numeri casuali, da cui andare a prelevare i numeri a caso, cioè inizializza la sequenza;
.. se esso è positivo non interessa il suo valore dato che fa solo proseguire nella sequenza; a parità del punto di partenza fornisce la stessa sequenza di numeri casuali;
.. se esso è zero il punto di partenza della sequenza viene influenzato dal valore del clock interno del calcolatore.

I numeri generati sono compresi tra 0 e 1 (minori di 1) e sono decimali. Per ottenere numeri interi in un dato intervallo si deve applicare un algoritmo.

Segue un programma esempio:

```
1 REM ES50 PROVA RND
5 OPEN4,4:CMD4
10 REM POSIZIONA INIZIO SEQUENZA
11 REM CON ARGOMENTO NEGATIVO RND
20 FORK=-5TO-1STEP1:PRINT
25 PRINT "INIZIO SEQUENZA CON RND(";K;")"
30 PRINTRND(K):GOSUB100
40 GOSUB400
45 GOSUB150
46 X=RND(0):PRINT"RND(0)=";X
47 X=RND(1):PRINT"RND(1)=";X
48 PRINT"RIPRISTINO BYTE 139/143"
50 GOSUB160:GOSUB400
55 NEXTK
60 PRINT#4:CLOSE4:STOP
100 FORL=0TO4:X(L)=PEEK(139+L):NEXTL
101 RETURN
150 FORL=0TO4:PRINT"BYTE(";139+L;")=";X(L)
151 NEXTL:RETURN
160 FORL=0TO4:POKE(139+L),X(L):NEXTL
161 RETURN
400 FORI=1TO2:FORJ=1TO3:PRINTRND(I*J);"  ";
401 NEXTJ:PRINT:NEXTI:RETURN
```

In esso, che manda i risultati su stampante per poterli esaminare più comodamente, facciamo variare un numero K tra -5 e -1, e, usandolo come argomento di RND, posizioniamo ogni volta il seme a un valore determinato. Poi vengono generati 6 numeri a caso con argomento positivo. Ogni volta che si fa girare il programma i numeri risultano uguali, infatti si ha sempre il posizionamento con lo stesso argomento negativo. Dopo aver posizionato il seme, con K negativo, memorizziamo in 5 variabili X(L) il contenuto dei 5 byte coinvolti dal sistema nella operazione RND (da 139 a 143). Poi stampiamo RND(0) e RND(1), che risultano ogni ciclo diversi. Alla linea 50, con il sottoprogramma 160, ripristiniamo i valori dei 5 byte da 139 a 143, con i valori che avevano al momento della RND(K) con K negativo, e stampiamo di nuovo 6 numeri casuali con argomento positivo. Come puoi vedere le due sequenze di numeri sono uguali ogni volta.

RISULTATI PROGRAMMA ES50

```
INIZIO SEQUENZA CON RND(-5 )
 3.73720468E-08
 .898884767    .930769958    .534739951
 .690061376    .718249081   3.80238951E-03
BYTE( 139 )= 104
BYTE( 140 )= 32
BYTE( 141 )= 131
BYTE( 142 )= 0
BYTE( 143 )= 0
RND(0)= .285157919
RND(1)= .458660293
RIPRISTINO BYTE 139/143
 .898884767    .930769958    .534739951
 .690061376    .718249081   3.80238951E-03

INIZIO SEQUENZA CON RND(-4 )
 2.99214662E-08
 .402343613    .92613015    .7171307
 .137023837    .770137118    .319573971
BYTE( 139 )= 104
BYTE( 140 )= 0
BYTE( 141 )= 131
BYTE( 142 )= 0
BYTE( 143 )= 0
RND(0)= .230470896
RND(1)= .431733826
```

```
RIPRISTINO BYTE 139/143
 .402343613   .92613015   .7171307
 .137023837   .770137118   .319573971

INIZIO SEQUENZA CON RND(-3 )
 4.48217179E-08
 .931279155   .556729296   .57212578
 .0406100041   .181847568   .172528894
BYTE( 139 )= 104
BYTE( 140 )= 64
BYTE( 141 )= 130
BYTE( 142 )= 0
BYTE( 143 )= 0
RND(0)= .148440719
RND(1)= .763706377
RIPRISTINO BYTE 139/143
 .931279155   .556729296   .57212578
 .0406100041   .181847568   .172528894

INIZIO SEQUENZA CON RND(-2 )
 2.99205567E-08
 .865554613   .846128798   .981100118
 .986081582   .514064711   .768773897
BYTE( 139 )= 104
BYTE( 140 )= 0
BYTE( 141 )= 130
BYTE( 142 )= 0
BYTE( 143 )= 0
RND(0)= .519533396
RND(1)= .747457131
RIPRISTINO BYTE 139/143
 .865554613   .846128798   .981100118
 .986081582   .514064711   .768773897

INIZIO SEQUENZA CON RND(-1 )
 2.99196472E-08
 .328780872   .978964086   .895758909
 .161031701   .0224078245   .0365292135
BYTE( 139 )= 104
BYTE( 140 )= 0
BYTE( 141 )= 129
BYTE( 142 )= 0
BYTE( 143 )= 0
```

```
RND(0)= 7.81309605E-03
RND(1)= .336002869
RIPRISTINO BYTE 139/143
 .328780872    .978964086    .895758909
 .161031701    .0224078245   .0365292135
```

Segue un semplice programma che lancia 20 volte un dado. Osserva alla linea 20 come si opera affinchè il numero sia compreso tra 1 e 6.

```
1 REM ES50BIS LANCIO DADI
5 PRINT" LANCIO 20 VOLTE UN DADO"
10 FORK=1TO4:FORJ=1TO5
20 X%=1+6*RND(K*J)
30 PRINTX%;" ";:NEXTJ:PRINT
40 NEXTK
50 STOP
```

RUN Tipo: Controllo

Formato: RUN [numero-linea]

dove numero-linea, se presente, deve essere un numero di linea esistente nel programma.

Questo comando è usato in modo immediato per far partire i programmi. Se manca il numero-linea, il programma parte dalla prima linea presente. Se numero-linea è presente il programma parte da quella linea. Prima di far partire il programma viene eseguito un CLR implicito di tutte le variabili.
Esso può essere usato anche dall'interno di un programma, che in questo caso fa ripartire se stesso, dalla linea citata o dall'inizio.

SAVE Tipo: Operazione I/O programmi

Formato: SAVE [fn] [,dn] [,ind]

dove i parametri hanno il solito significato precisato all'inizio di questo paragrafo.

Serve per memorizzare programmi su nastro o su disco. Di norma si usa il nome di un file per memorizzare il programma, questo per poterlo richiamare poi con un nome; comunque il nome può anche essere tralasciato quando si memorizza su nastro, deve essere presente se si usa il disco. Lo fn può essere una variabile stringa o

una stringa. Il sistema attribuisce al file il tipo PRG, che significa PROGRAMMA. Se viene omesso dn, esso è assunto uguale a 1 e quindi la memorizzazione avviene su nastro. Per il disco si deve porre dn=8. Dopo il salvataggio del programma esso è ancora in memoria e può continuare ad operare. Se il comando è dato dall'interno di un programma, esso salva se stesso e poi prosegue con l'istruzione successiva. Il terzo parametro, ind, può essere uguale a 1 per memorizzare il programma, mantenendo, al momento del LOAD, invariato il punto di inizio in memoria (diverso dal normale 2049). Per il nastro si può usare ind=2 per far memorizzare una segnalazione di fine-nastro dopo il programma; ind=3 per ottenere i due effetti combinati, cioè indirizzo di caricamento invariato e segnalazione di fine-nastro.

Esempi del comando:

SAVE	memorizza su nastro senza nome
SAVE "PRO1"	memorizza su nastro con nome PRO1
SAVE "PRO1",8	memorizza su disco con nome PRO1
SAVE "PRO1",1,1	memorizza su nastro mantenendo l'indirizzo di inizio
SAVE "PRO1",1,2	memorizza su nastro con segnalazione di fine-nastro
SAVE "PRO1",1,3	memorizza mantenendo l'indirizzo di inizio e con segnalazione di fine-nastro
SAVE "PRO1",8,1	memorizza su disco mantenendo l'indirizzo di inizio.

SGN Tipo: Funzione matematica

Formato: SGN (espressione)

dove espressione deve essere numerica.

Fornisce un risultato intero:

.. 1 se il valore dell'argomento è positivo,
.. 0 se il valore dell'argomento è zero,
..-1 se il valore dell'argomento è negativo.

Segue un semplice esempio:

```
1 REM ES53 PROVA SGN
10 INPUT"NUMERO: ";N
20 PRINT"SGN(";N;")=";SGN(N);
30 ON(SGN(N)+2)GOTO40,50,60
40 PRINT" NEGATIVO":STOP
50 PRINT" NULLO":STOP
60 PRINT" POSITIVO":STOP
```

SIN Tipo: Funzione matematica

Formato: SIN (numero)

dove numero può anche essere una variabile numerica o un'espressione numerica.

L'argomento deve essere espresso in radianti. Segue un semplice esempio:

```
1 REM ES54 PROVA SIN
10 INPUT"ARGOMENTO IN RADIANTI: ";N
20 PRINT"SIN(";N;")=";SIN(N)
30 PRINT"COS(";N;")=";SIN(N+π/2)
40 STOP
```

SPC Tipo: Funzione stampa

Formato: SPC (espressione)

dove espressione deve dare un valore numerico compreso tra 0 e 255 per stampante e nastro, tra 0 e 254 per il disco.

Se il valore di espressione non è intero, i decimali vengono troncati. La funzione provoca lo stesso effetto della stampa di una stringa di N spazi, se N è il valore dell'espressione. Se diretta alla stampante, quando provoca uno spazio nell'ultima posizione della linea si ha un passaggio automatico alla linea successiva, con eventuale perdita degli spazi residui.

Segue un esempio di uso sul video.

```
1 REM ES55 PROVA SPC
10 A$="PRIMA STRINGA"
15 B$="SECONDA STRINGA"
20 PRINTA$;SPC(7);B$
```

```
25 PRINTA$SPC(7)B$
30 A1$="BELLO":A2$="BELLISSIMO"
35 A3$="PIACEVOLE":A4$="STUPENDO"
40 PRINT"M  "A1$SPC(20-LEN(A1$))A2$
45 PRINT"  "A3$SPC(20-LEN(A3$))A4$
50 STOP
```

Nota l'uso sofisticato di SPC, usando nell'argomento la lunghezza della stringa appena stampata, per ottenere gli incolonnamenti. Segue un risultato su stampante del programma, ottenuto scrivendo in immediato:

OPEN4,4:CMD4:GOTO1

come puoi notare il carattere di controllo cursore-giù incorporato nella stringa di inizio della linea 40 ha sulla stampante l'effetto di cambiare il set di caratteri.

RISULTATI SU STAMPANTE DI ES55

```
PRIMA STRINGA          SECONDA STRINGA
PRIMA STRINGA          SECONDA STRINGA
  bello                  bellissimo
  piacevole              stupendo


break in 50
```

SQR Tipo: Funzione matematica

Formato: SQR (espressione)

dove espressione deve essere un'espressione numerica con valore positivo.

Fornisce la radice quadrata dell'argomento.

Segue un esempio.

```
1 REM ES56 PROVA SQR
10 INPUT"NUMERO: ";N
20 IFN<0THENPRINT"NUMERO NEGATIVO":GOTO10
30 PRINT"SQR(";N;")=";SQR(N)
40 STOP
```

STATUS Tipo: Funzione lettura memoria

Formato: STATUS

Fornisce un valore numerico intero che informa sull'esito dell'ultima operazione di I/O eseguita su un file. Si può usare per qualunque periferica sulla quale si è aperto un file. Si abbrevia in ST.
I valori di ST hanno il seguente significato:

Pos.bit	Valore decim.	Lett. nastro	Verify/Load nastro	Bus ser. I/O
0	1			fuori tempo scritt.
1	2			fuori tempo lett.
2	4	blocco corto		
3	8	blocco lungo		
4	16	err.lett.	errore	
5	32	errore checksum		
6	64	fine file		EOI
7	—128	fine nastro		device non presente

Quando ST è zero significa che l'operazione è andata bene, se diverso da zero, si deve controllare il valore per stabilire cosa fare. Il valore 64 e il valore -128 non sono errori, ma solo utili indicazioni per il programma. Devi fare attenzione e memorizzare ST in una variabile di comodo, se lo analizzi in un passo di programma successivo, infatti ogni operazione file interviene su ST. Tipico esempio è la lista di un file sequenziale letto da nastro; quando leggi l'ultimo dato, ST diventa 64, se non lo memorizzi e vai a stampare (la stampante è aperta come file), ST cambia, quando vai ad analizzare la segnalazione per vedere se è finito il file non trovi più il valore 64.

STEP Tipo: Controllo

Formato: STEP espressione

dove espressione deve essre numerica, positiva o negativa.

Può essere presente in una frase FOR dopo il valore finale della variabile di controllo del ciclo. Viene usato ogni ciclo per modificare il contatore del ciclo. Non ha senso porlo uguale a zero, perchè produce un ciclo infinito.
Vai a rivedere la descrizione di FOR.

STOP Tipo: Controllo

Formato: STOP

Ferma il programma con lo stesso effetto di quando si preme il tasto RUN/STOP. Compare il messaggio: ?BREAK IN LINE .... Per far proseguire il programma si deve usare il comando CONT. E' molto utile per verificare particolari risultati durante le prove di un programma; dopo gli STOP possono essere eliminati.

STR$ Tipo: Funzione stringa

Formato: STR$ (espressione)

dove espressione deve essere numerica.

Trasforma il valore numerico di espressione in stringa. Accetta come argomento un numero intero o decimale con segno.

Segue un esempio:

```
1 REM ES60 PROVA STR$
5 M$="LUNGHEZZA STRINGA GENERATA: "
10 INPUT"NUMERO: ";N
20 PRINT"STR$(";N;")=";STR$(N);"*"
25 PRINTM$;LEN(STR$(N))
27 PRINTN;"*"
29 PRINTSTR$(N);"*"
30 STOP
```

RISULTATI ES60

```
NUMERO:
STR$( 5678 )= 5678*
LUNGHEZZA STRINGA GENERATA:  5
 5678 *
 5678*


STR$(-34567878 )=-34567878*
LUNGHEZZA STRINGA GENERATA:  9
-34567878 *
-34567878*
```

```
STR$( 4.5678E-09 )= 4.5678E-09*
LUNGHEZZA STRINGA GENERATA:  11
 4.5678E-09 *
 4.5678E-09*
```

Come puoi vedere dai risultati dell'esempio la stringa ottenuta dalla funzione ha un carattere meno del numero in formato stampa, manca lo spazio dopo il numero; abbiamo usato l'artificio di stampare un asterisco a chiusura della stampa. Come vedi si possono scrivere i numeri in qualunque formato.

SYS Tipo: Controllo

Formato: SYS indirizzo

dove indirizzo è un indirizzo di memoria.

Questo comando serve per andare ad eseguire un programma scritto in linguaggio macchina e memorizzato a partire dall'indirizzo citato. Indirizzo può anche essere un'espressione numerica, purchè fornisca un numero valido, tra 0 e 65535. Il programma in linguaggio maccchina deve terminare con il comando RTS, che fa proseguire il programma Basic dall'istruzione seguente la SYS. Il codice decimale di RTS è 96, puoi provare a scrivere:

POKE4400,96:SYS4400

fa andare a 4400, dove trova RTS e quindi torna al Basic.

Il comando SYS 64738 ripristina le condizioni iniziali del calcolatore al momento dell'accensione, mandando in esecuzione una routine del sistema operativo.

TAB Tipo: Funzione stampa

Formato: TAB (espressione)

dove espressione deve fornire un valore numerico compreso tra 0 e 255.

Il valore di espressione fornisce il numero di posizioni di stampa da contare partendo dall'inizio della linea attuale per arrivare alla posizione di stampa, quando è usato in una istruzione PRINT diretta al video. Quando invece TAB compare in una istruzione di PRINT diretta alla stampante, l'argomento rappresenta il numero di spazi da lasciare prima della prossima stampa, partendo dalla posizione attuale, cioè si comporta come SPC. Segue un programma esempio, che abbiamo ricavato modificando l'esempio riportato per SPC.

```
1 REM ES62 PROVA TAB
10 A$="PRIMA STRINGA"
15 B$="SECONDA STRINGA"
20 PRINTA$;TAB(15);B$
25 PRINTA$TAB(15)B$
30 A1$="BELLO":A2$="BELLISSIMO"
35 A3$="PIACEVOLE":A4$="STUPENDO"
40 PRINT"▣  "A1$TAB(20)A2$
45 PRINT"   "A3$TAB(20)A4$
50 STOP
```

TAN Tipo: Funzione matematica

Formato: TAN (espressione)

dove espressione deve dare un valore numerico in radianti dell'angolo di cui si vuole la tangente.

Segue un esempio:

```
1 REM ES63 PROVA TAN
10 INPUT"ARGOMENTO IN RADIANTI: ";N
20 PRINT"TAN(";N;")=";TAN(N)
50 STOP
```

THEN Tipo: Controllo

Formato: THEN numero linea
THEN istruzione

è collegata alla parola chiave IF alla quale rimandiamo.

TIME Tipo: Funzione lettura memoria

Formato: TI

fornisce il contenuto del clock interno in sessantesimi di secondo. Questo numero diviso 60 fornisce i secondi trascorsi dall'accensione del calcolatore, o dal ripristino delle condizioni iniziali, o dall'azzeramento di TI, che però non può essere ottenuto direttamente, ma azzerando la stringa TI$.

Segue un esempio, che tratta sia TI che TI$.

```
1 REM ES65 PROVA TIME
10 PRINT"SEC. DALL'ACCENSIONE ";
15 PRINTTI/60
20 PRINT"TI$=";TI$
30 TI$="000000"
35 PRINT"SEC. DOPO AZZERAMENTO TI$ ";
40 PRINTTI/60
60 REM CALCOLO INTERVALLO TEMPO
61 FORI=1TO4
70 INPUT"LIMITE CICLO FOR: ";N
72 A=TI
75 FORK=0TON:NEXTK
77 B=TI
79 PRINT"INTERVALLO: ";(B-A)/60
80 NEXTI:STOP
```

Come puoi vedere, viene inizialmente stampato il contenuto di TI/60 e di TI$. La stringa contenuta in TI$ va letta: hhmmss, cioè ore, minuti, secondi.
Alla linea 30 viene azzerato TI$ e poi viene stampato di nuovo il contenuto di TI/60. Poi, per 4 volte, viene chiesto un limite N per il ciclo FOR e viene calcolato l'intervallo di tempo che trascorre per eseguire FOR...NEXT N volte.

TIME$ Tipo: Funzione lettura/scrittura memoria

Formato TI$

Vedi l'esempio precedente.

TO Tipo: Controllo

Formato: TO limite

Fa parte del ciclo FOR; rimandiamo alle spiegazioni relative.

USR Tipo: Funzione controllo

Formato: USR (espressione)

dove espressione deve essere numerica.

Serve per andare ad eseguire un programma in linguaggio macchina, il cui indirizzo deve essere stato preventivamente memorizzato nei byte 785 e 786, con delle

istruzioni POKE. Il valore di espressione viene memorizzato nell'accumulatore floating-point che inizia al byte 97; in tale accumulatore si trova poi il risultato del calcolo eseguito. Il risultato comunque viene reso disponibile al programma Basic, come valore della funzione. Il programma in linguaggio macchina deve terminare con l'istruzione RTS.

VAL Tipo: Funzione conversione

Formato: VAL (stringa numerica)

dove stringa numerica deve essere un qualunque numero scritto come stringa.

La funzione trasforma il numero in stringa. Se la stringa ha il primo carattere diverso da spazio che non è un segno o una cifra, il valore fornito è zero. La conversione si ferma al primo carattere non numerico incontrato, senza segnalare errore.

Segue un esempio:

```
1 REM ES69 PROVA VAL
10 INPUT"STRINGA NUMERICA: ";N$
20 PRINT"VAL(";N$;")=";VAL(N$)
30 STOP
```

VERIFY Tipo: Operazione I/O programmi

Formato: VERIFY [fn] [,dn]

dove i parametri hanno il solito significato.

Può essere usata in modo immediato o da programma. Esegue la verifica di un file programma confrontandolo con quello presente in memoria. E' buona norma verificare sempre le registrazioni effettuate. Se si omette dn, il sistema assume si tratti del nastro. Sempre nel caso del nastro se si omette il nome, viene verificato il primo programma incontrato; il nastro deve essere ovviamente correttamente posizionato. Per il disco è obbligatorio il nome del file.
Il nome del programma può essere una stringa tra virgolette o una variabile, o una espressione stringa.
Durante la verifica si svolge un colloquio tramite video. Nel caso del nastro viene chiesto di premere il tasto PLAY di avviamento.

WAIT Tipo: Controllo

Formato: WAIT indirizzo, maschera-1, [maschera-2]

dove: indirizzo è un indirizzo di memoria, cioè un byte che si desidera analizzare; maschera-1, deve essere un numero compreso tra 0 e 255, viene usato per operare un AND con il risultato dell'operazione di OR-esclusivo (XOR) tra il contenuto di indirizzo e maschera-2. Questa maschera estrae dal risultato i bit 1 corrispondenti ai suoi bit 1;
maschera-2, deve essere un numero compreso tra 0 e 255, viene usato per operare un OR-esclusivo con il contenuto dell'indirizzo; esso fornisce bit 0 dove trova bit uguali e bit 1 dove trova bit diversi.
Se maschera-2 manca, esso viene assunto uguale a zero.

Le regole dell'operazione di OR-esclusivo sono le seguenti:

1 XOR 1 = 0
1 XOR 0 = 1
0 XOR 1 = 1
0 XOR 0 = 0

Tieni presente che con WAIT puoi entrare in un ciclo infinito. Segue un programma che puoi provare; in caso per uscire puoi tenere premuto il tasto RUN/STOP e premere RESTORE.

```
1 REM ES71 PROVA WAIT
10 INPUT"PARAMETRI WAIT: ";X,Y,Z
15 A=TI
16 WAITX,Y,Z
17 B=TI
20 PRINT"WAIT TERMINATO"
25 PRINT"INTERVALLO SEC.: ";(B-A)/60
30 STOP
```

Il programma prosegue se il risultato della operazione logica sul contenuto dell'indirizzo risulta vero, cioè diverso da zero, mentre torna all'operazione di WAIT se il risultato risulta falso, cioè uguale a zero.

Se provi con X=162, che è il byte basso del clock, e con diversi valori per Y e Z puoi ottenere alternativamente cicli infiniti o intervalli di riposo più o meno lunghi.

Puoi provare anche il programma ES71BIS, che segue, dove vengono chiesti,

l'indirizzo X di un byte e le due maschere, poi viene chiesto il valore da scrivere nel byte X e viene lanciata l'istruzione WAIT. Puoi usare per X l'indirizzo di un byte in RAM situato dopo il programma; se usi 4000 sei sicuro di andar bene. Puoi provare a scrivere in 4000 diversi valori mantenendo fisse le maschere. Puoi usare per X anche 162, che è il byte basso del clock e quindi si modifica da solo, e inizializzarlo con un numero scelto da te.

```
1 REM ES71BIS PROVA WAIT
10 INPUT"PARAMETRI WAIT: ";X,Y,Z
13 PRINT"VALORE PER BYTE";X;": ";:INPUTN
15 PRINT:POKEX,N:A=TI
16 WAITX,Y,Z
17 B=TI
20 PRINT"WAIT TERMINATO"
23 PRINT"CONTENUTO X: ";PEEK(X)
25 PRINT"INTERVALLO SEC.: ";(B-A)/60
30 STOP
```

I due programmi ti fanno vedere la durata approssimata del ciclo WAIT. Il programma ES71BIS stampa anche il contenuto di X dopo WAIT; esso sarà il contenuto iniziale se il byte non viene automaticamente modificato dal sistema in tempo reale, come il clock.

Esempi numerici delle operazioni XOR e AND:

1)

Cont.Ind.	Maschera-1	Maschera-2
3	3	255
00000011	00000011	11111111
XOR tra	00000011 e	
	11111111 dà	11111100
AND tra	11111100 e	
	00000011 dà	00000000

producendo un ciclo di attesa infinito.

2)

3	7	255
00000011	00000111	11111111
XOR tra	00000011 e	
	11111111 dà	11111100
AND tra	11111100 e	
	00000111 dà	00000100

e il ciclo termina subito.

3)

3	255	0
00000011	11111111	00000000
XOR tra	00000011 e	
	00000000 dà	00000011
AND tra	00000011 e	
	11111111 dà	00000011

e il ciclo termina subito.

CAPITOLO 3

COLLOQUIO VIDEO/TASTIERA

3.1 I DUE STATI DEL CALCOLATORE

Quando accendi il tuo COMMODORE 64 e compare la scritta iniziale è in funzione il programma EDITOR e il calcolatore si trova nello STATO SISTEMA.

I due stati possibili per il calcolatore sono:
.. STATO SISTEMA;
.. STATO PROGRAMMA.

Siamo nel primo stato quando comunichiamo con il sistema calcolatore impartendo comandi tramite la tastiera. Siamo nel secondo stato quando sta funzionando un programma in Basic caricato nella memoria RAM.
Quando il programma Basic si interrompe per uno dei seguenti motivi:
.. abbiamo premuto il tasto RUN/STOP;
.. il programma ha incontrato uno STOP o un END oppure ha eseguito l'ultima istruzione fisicamente presente;
.. il sistema ha interrotto il programma segnalando con un messaggio la presenza di un errore;
si passa dallo stato programma allo stato sistema. Quando vengono eseguite istruzioni Basic in modo immediato, si transita dallo stato programma per il tempo necessario all'esecuzione delle istruzioni, e si torna allo stato sistema.
Un programma Basic può far rimanere indefinitamente il calcolatore nello stato programma; per esempio se contiene un ciclo infinito, oppure se si mette in attesa della pressione di un particolare tasto (GET) e questo non viene premuto. Quando il programma Basic è in attesa di ricevere dati con una istruzione INPUT non viene sentito il tasto RUN/STOP; dopo vari tentativi si può riuscire ad interrompere il programma premendo contemporaneamente RUN/STOP e RESTORE.

3.2 USO TASTI E CARATTERI DI CONTROLLO

Nel Paragrafo 1.2 abbiamo visto il significato dei tasti che permettono di muoversi sul video; li riassumiamo qui. Nella descrizione che segue i termini "viene visualiz-

zato" o "appare" si riferiscono all'effetto dei tasti quando si sta scrivendo dopo avere "aperto le virgolette" (quote mode).

CLR/HOME

viene visualizzato come una S in campo inverso, manda il cursore nell'angolo in alto a sinistra del video, senza cancellare il suo contenuto. Corrisponde a CHR$(19).

SHIFT + CLR/HOME

viene visualizzato come un cuore in campo inverso, pulisce il video e manda il cursore nell'angolo in alto a sinistra. Corrisponde a CHR$(147).

CRSR/frecce-verticali

viene visualizzato come Q in campo inverso, sposta il cursore di una posizione verso il basso. Corrisponde a CHR$(17).

SHIFT + CRSR/frecce-verticali

viene visualizzato come un cerchio in campo inverso, sposta il cursore di una posizione verso l'alto. Corrisponde a CHR$(145).

CRSR/frecce-orizzontali

viene visualizzato come una parentesi quadra destra in campo inverso, sposta il cursore di una posizione verso destra. Corrisponde a CHR$(29).

SHIFT + CRSR/frecce-orizzontali

viene visualizzato come una barretta verticale in campo inverso, sposta il cursore di una posizione verso sinistra. Corrisponde a CHR$(157).

Per modificare i dati presenti sul quadro video si deve portare il cursore nella posizione desiderata e, poi, o riscrivere sopra o inserire o cancellare caratteri. I tasti per inserire o cancellare sono:

INST/DEL

cancella il carattere a sinistra del cursore e sposta il cursore e tutto quello che lo segue verso sinistra di una posizione. Corrisponde a CHR$(20).

SHIFT + INST/DEL

crea uno spazio a sinistra del cursore, spostando il cursore e tutto quello che lo segue di una posizione verso destra. Corrisponde a CHR$(148).

Per passare alla visualizzazione dei caratteri in campo inverso si devono usare contemporaneamente i due tasti CTRL e 9, che appare come una R in campo inverso e corrisponde a CHR$(18). Per tornare al modo normale si devono premere contemporaneamente i due tasti CTRL e 0, che appare come una barrettina orizzontale in campo inverso e corrisponde a CHR$(146).

Devi dedicare una particolare attenzione al video quando apri le virgolette per scrivere una stringa. I tasti di controllo possono essere usati all'interno di una stringa e appaiono con il carattere descritto. Quelli che creano problemi sono INST/DEL e SHIFT + INST/DEL, e quindi è meglio non usarli.
Se devi andare a modificare qualcosa all'interno di una stringa ti conviene uscire con il tasto RETURN, e poi posizionarti spostando il cursore all'interno della stringa e usare nel solito modo i tasti di cancelazione e inserzione.
I tasti di controllo, invece di comparire all'interno di una stringa, possono essere creati usando la funzione CHR$ avente come argomento il codice ASCII corrispondente.
Ricorda che nei listati su stampante dei programmi i caratteri di controllo contenuti all'interno delle stringhe possono apparire diversamente che sul video; questo dipende dalle caratteristiche della stampante.

Le caratteristiche dell'EDITOR consentono di scrivere facilmente i programmi in Basic. Sul video appaiono le linee che tu scrivi; se le scrivi in disordine basta che tu usi il comando LIST seguito dai possibili parametri per avere in ordine il listato delle parti di programma che ti interessano. Per correggere linee di programma puoi portarti con il cursore sulla posizione desiderata e correggere; quando premi RETURN la correzione viene accettata.
Puoi cambiare il numero di linea, però ricordati di abolire la linea precedente. Per abolire una linea di programma basta scrivere il suo numero e premere RETURN.
Puoi ripetere linee di programma, che sono poco diverse tra loro, andando a cambiare il numero di linea, poi puoi listarle e apportare nelle copie le necessarie modifiche.
Questa ricchezza di prestazioni presenta anche dei pericoli; infatti si possono produrre modifiche non desiderate. Dopo una modifica ti conviene sempre listare il pezzo di programma coinvolto e controllare.

3.3 CONTROLLO COLORE

I byte che controllano il colore quando si usa il calcolatore in modo normale (no grafica, no multicolor) sono i seguenti:

.. 53280 colore bordo (codice colore +240);
.. 53281 colore sfondo (codice colore +240);
.. 646 colore caratteri (codice colore).

Al momento dell'accensione del calcolatore la situazione è la seguente:

.. 53280 contiene 254 (codice colore 14 + 240);
.. 53281 contiene 246 (codice colore 6 + 240);
.. 646 contiene 14 (codice colore 14);

il colore del cursore e il colore del bordo sono uguali, azzurro, mentre il colore dello sfondo è blu.

Sotto EDITOR, se premi contemporaneamente il tasto CTRL e un tasto numerico da 1 a 8 oppure il tasto COMMODORE e un tasto numerico da 1 a 8, vedi cambiare il colore del cursore; quello che scrivi dopo appare nel colore del cursore.
Puoi incorporare in una stringa i caratteri speciali che corrispondono alla pressione contemporanea di queste coppie di tasti; quando la stringa viene stampata si ha lo stesso effetto di cambio colore.
Puoi scrivere con POKE nel byte 646 i codici ASCII che corrispondono a queste coppie di tasti e ottieni lo stesso effetto di cambio colore.
I codici colore, cioè i numeri corrispondenti ai colori, non corrispondono ai tasti numerici che si usano; si ha che:
..premendo i tasti numerici contemporaneamente a CTRL, al tasto 1 corrisponde il numero colore 0 e, andando in sequenza, al tasto 8 il numero colore 7;
..premendo i tasti numerici contemporaneamente a COMMODORE, al tasto 1 corrisponde il numero colore 8 e, andando in sequenza, al tasto 8 il numero colore 15.

Segue il programma COLORI1, nel quale puoi vedere tutte queste cose. Esso opera così:
.. pone nella stringa A$, di 16 posizioni, i caratteri che si generano premendo contemporaneamente CTRL e i tasti da 1 a 8 e COMMODORE e i tasti da 1 a 8;
.. prepara con dei DATA i nomi dei 16 colori;
.. con un FOR che esegue 16 cicli, preleva dalla stringa A$ i caratteri-colore in sequenza, li stampa uno per volta, ottenendo di modificare il colore del cursore e quindi dei caratteri, e va a leggere e memorizzare nella matrice C% di 16 righe e 3 colonne i contenuti dei byte 53280, 53281 e 646;

.. apre la stampante e stampa una tabellina, nella quale sono riportati i nomi dei colori, i contenuti dei tre byte citati, il carattere corrispondente al colore come appare tra le virgolette e il codice ASCII corrispondente (quello da usare come argomento della funzione CHR$).

```
1 REM COLORI1
5 DIMC%(15,2),C$(15):S$="                            "
6 M$="COLORE         53280 53281 646    CAR. CHR$"
10 A$="[illegible]"
11 DATANERO,BIANCO,ROSSO, CIANO,PORPORA
12 DATAVERDE,BLU,GIALLO,ARANCIONE, MARRONE
13 DATAROSSO-CHIARO,GRIGIO-1,GRIGIO-2
14 DATAVERDE-CHIARO,AZZURRO,GRIGIO-3
15 FORK=0TO15
20 C$(K)=MID$(A$,K+1,1):PRINTC$(K);
25 C%(K,0)=PEEK(53280)-240:REM COLORE BORDO
30 C%(K,1)=PEEK(53281)-240:REM COLORE SFONDO
35 C%(K,2)=PEEK(646):REM COLORE CURSORE
40 NEXTK:PRINT
43 OPEN4,4:CMD4
45 PRINT"           CONTENUTI BYTE COLORI"
50 PRINT:PRINTM$:PRINT
55 FORK=0TO15:READC$:B$=LEFT$(C$+S$,13)
57 X$=STR$(C%(K,0)):X$=LEFT$(X$+S$,5)
58 Y$=STR$(C%(K,1)):Y$=LEFT$(Y$+S$,6)
60 PRINTB$;X$;Y$;
61 X$=STR$(C%(K,2)):X$=LEFT$(X$+S$,6)
63 PRINTX$;
65 PRINTCHR$(34)C$(K)CHR$(34);
67 PRINT" ";ASC(C$(K))
70 NEXTK:PRINT#4:CLOSE4
```

RISULTATI PROGRAMMA COLORI

CONTENUTI BYTE COLORI

COLORE	53280	53281	646	CAR.	CHR$
NERO	14	6	0	"█"	144
BIANCO	14	6	1	"█"	5
ROSSO	14	6	2	"█"	28
CIANO	14	6	3	"█"	159
PORPORA	14	6	4	"█"	156
VERDE	14	6	5	"█"	30
BLU	14	6	6	"█"	31
GIALLO	14	6	7	"█"	158
ARANCIONE	14	6	8	"█"	129
MARRONE	14	6	9	"█"	149
ROSSO-CHIARO	14	6	10	"█"	150
GRIGIO-1	14	6	11	"█"	151
GRIGIO-2	14	6	12	"█"	152
VERDE-CHIARO	14	6	13	"█"	153
AZZURRO	14	6	14	"█"	154
GRIGIO-3	14	6	15	"█"	155

Consultando la tabellina, trovi naturalmente che il contenuto dei byte 53280 e 53281 resta sempre uguale, infatti la PRINT del carattere-colore non modifica sfondo e bordo, ma solo il colore del cursore.

Abbiamo preparato il programma COLORI2, per provare le combinazioni di colore e trovare quelle che ci soddisfano.

```
1 REM COLORI2
6 PRINT"█**CAMBIO COLORI**██"
7 DATANERO,BIANCO,ROSSO, CIANO,PORPORA
8 DATAVERDE,BLU,GIALLO,ARANCIONE, MARRONE
9 DATAROSSO-CHIARO,GRIGIO-1,GRIGIO-2
10 DATAVERDE-CHIARO,AZZURRO,GRIGIO-3
11 DIMD$(15):FORK=0TO15
12 READD$(K):NEXTK
15 INPUT"███COLORE BORDO:       ";B%
20 INPUT"COLORE SFONDO:      ";S%
25 INPUT"COLORE CARATTERE:  ";C%
```

```
26 X$=D$(B%):Y$=D$(S%):Z$=D$(C%)
27 POKE53280,B%OR240
28 POKE53281,S%OR240
29 POKE646,C%
30 PRINT"[illegible]BORDO: "X$;" (";B%;")"
35 PRINT"[illegible]SFONDO: "Y$;" (";S%;")"
40 PRINT"CARATTERI: "Z$;" (";C%;")"
41 PRINT"BYTE 646= ";PEEK(646)
44 PRINT"BYTE 53281= ";PEEK(53281)
45 PRINT"BYTE 53280= ";PEEK(53280)
50 PRINT"[illegible]PREMI UN TASTO PER CONTINUARE"
55 GETB$:IFB$=""THEN55
60 GOTO15
```

In esso ti viene chiesto il codice colore per bordo, sfondo e carattere (cursore), poi vengono modificati i contenuti dei tre byte relativi con delle istruzioni POKE e vengono stampati i nomi dei colori scelti, i numeri-colore corrispondenti e i contenuti dei tre byte. Alla linea 55 abbiamo messo un ciclo di attesa, per interromperlo devi premere un qualunque tasto e proseguire.
Vedrai che se tra una prova e l'altra usi come colore dello sfondo il codice che hai usato precedentemente per i caratteri le 3 prime righe di stampa scompaiono dal video, ànche se esistono ancora, infatti se il colore dello sfondo e quello dei caratteri coincidono non si può leggere.
Vedrai anche che queste prime 3 righe dopo i cambi dei colori rimangono di un colore diverso, infatti nella mappa colori del video il sistema pone il colore nel momento che scrive un carattere e non lo modifica se cambia il colore del cursore. Se non fosse così non si potrebbero ottenere quadri video con tanti colori diversi.

Da quanto abbiamo visto, il sistema dispone di 16 colori, essi possono essere usati sia per il bordo, che per lo sfondo, che per i caratteri.
Ne' volume dedicato alla grafica approfondiremo anche l'argomento colore.

Per poter vedere tutti insieme i 16 colori disponibili abbiamo preparato il programma COLORI3; in esso vengono stampati sul video 16 rettangoli ognuno di un colore diverso. Quando il colore del carattere uguaglia il colore dello sfondo non si vede il carattere.

```
1 REM COLORI3
3 DIMC$(15):B$="[illegible]":D$="":PRINT"[illegible]";
4 FORK=0TO9:D$=D$+B$:NEXTK
5 A$="[illegible]":POKE53281,241
6 FORK=0TO15:C$(K)=MID$(A$,K+1,1):NEXTK
```

```
10 FORK=0TO15STEP4
15 FORL=0TO4
20 FORJ=0TO3
30 PRINTC$(K+J);D$;
35 NEXTJ:NEXTL
40 NEXTK
```

Per ottenere il codice del colore usiamo la stringa già usata nei programmi precedenti. Per fare risaltare i colori usiamo il colore bianco per lo sfondo.

La mappa colore inizia in 55296 ed occupa 1000 byte. Segue il programma COLORI4, che stampa sul video 16 righe ognuna di un colore diverso andando a scrivere con POKE il codice di un carattere nella mapppa video e il codice del colore nella corrispondente posizione della mappa colori. Questa operazione viene fatta 16 volte, cambiando ogni volta il colore dello sfondo. Ogni volta che il quadro video è completo c'è un ciclo di attesa e poi inizia la preparazione del quadro seguente.

```
1 REM COLORI4
3 B$="●":PRINT"[illegible]"
4 D=ASC(B$)
5 FORK=0TO639:POKE1024+K,D:NEXTK
6 FORM=240TO255
7 POKE53281,M
9 MV=1024:MC=55296
13 FORK=0TO15
15 FORL=0TO39
30 POKEMC+L,K
37 NEXTL:MV=MV+40:MC=MC+40
40 NEXTK
50 FORI=1TO600:NEXTI
55 NEXTM
```

Alla linea 3 abbiamo posto in B$ un carattere, e stampato una stringa di caratteri di controllo per mandare il cursore nella parte bassa del video. Alla linea 4 abbiamo calcolato il codice ASCII del carattere posto in B$. Alla linea 5 abbiamo posto nella mappa video 640 (16x40) volte il carattere di B$. Tale carattere non risulta visibile dato che la mappa colore contiene i codici del colore dello sfondo. Alla linea 6 abbiamo istituito un ciclo che fa cambiare nei 16 modi possibili il colore dello sfondo. Alla linea 7 abbiamo modificato il colore dello sfondo. Alla linea 9 abbiamo inizializzato i puntatori alle mappe video e colore. Dalla linea 13 alla linea 40 abbiamo sviluppato due cicli concatenati per mettere in ognuna delle 16 linee

video già occupate dai caratteri il codice colore e far comparire i caratteri. Il codice colore viene messo nelle posizioni della mappa colore, cambiando colore ogni 40 posizioni. Alla linea 50 abbiamo posto un ciclo d'attesa prima di tornare in 7 per efffetto del NEXT della linea 55 e ricominciare con un nuovo colore dello sfondo.

3.4 INPUT CONTROLLATO

Con le parole "input controllato" intendiamo un metodo per ricevere dati di Input dalla tastiera senza possibilità di errore, entro i limiti che precisiamo.

Quando, sia con la istruzione INPUT che con la istruzione GET, usiamo una variabile stringa per ricevere dati, vengono accettati tutti i caratteri. Questo è completamente vero se prima di rispondere alla richiesta di dati apriamo le virgolette e le chiudiamo prima di premere il tasto RETURN.
All'interno delle virgolette sono accettati anche i separatori virgola e due punti. Se non apriamo le virgolette, i caratteri virgola o due punti, sono recepiti come la fine di un campo dato e l'inizio del successivo; per questa ragione è probabile che scriviamo un numero di dati superiore al numero delle variabili disponibili nella lista di ingresso e quindi vedìamo comparire sul video il messaggio: EXTRA IGNORED. In questo caso il programma prosegue, ma il video diventa poco gradevole alla vista.
Quando, invece, si usano variabili numeriche, se nei dati è presente anche un solo carattere non valido, compare il messaggio: REDO FROM START e viene chiesto nuovamente il dato (o i dati). Anche in questo caso il video non risulta gradevole. Inoltre, se abbiamo previsto di fare entrare in un quadro video un numero di righe abbastanza vicino alla capienza, pensando ogni dato non molto lungo, qualora un dato sia più lungo del previsto, il quadro video si rovina.

Da quanto detto emerge che per una buona gestione del quadro video dovremmo controllare i dati di ingresso per il tipo dei caratteri e per la lunghezza dei campi. I limiti al controllo di errore dipendono dal fatto che se invece di scrivere una parola ne scriviamo un'altra la cosa non può essere controllata e lo stesso discorso vale per i numeri.

Il controllo più rigoroso per i dati si può avere ricevendoli carattere per carattere con l'istruzione GET seguita da una variabile stringa, ma il metodo risulta un pò lento. In questo caso però si riesce a controllare ogni singolo carattere e la lunghezza del dato, e non si rischia di rovinare il quadro video.

Un altro metodo può essere quello di riservare una zona video per ricevere i dati nella parte bassa usando l'istruzione INPUT seguita da una variabile stringa; lasciare il quadro video sopra e trasferirvi il dato nella giusta posizione, dopo averne controllato la validità.

Segue il programma INPUT1, nel quale si controlla un dato numerico letto con GET.

```
1 REM INPUT1
10 REM CONTROLLO DATO NUMERICO
15 REM LETTO CON GET
20 REM NOTA LA LUNGHEZZA MASSIMA DEL DATO
25 INPUT"██QUANTI CARATTERI: ";L
26 IFL=0THENSTOP
27 IFL<0THENPRINT"]]":GOTO25
29 PRINT"SCRIVI DATO":GOSUB500
30 IFE=0THENPRINT"██ACCETTATO: "N$:GOTO25
35 PRINT"██NON VALIDO":GOTO25
500 N=0:N$="":E=0:P=0:S=0:B$=""
503 PRINT"█DATO: ";
505 FORK=1TOL
510 A$="":GETA$
515 IFA$=""THEN510
517 IFA$=CHR$(13)THEN553
520 IFA$>"/"ANDA$<":"THEN550
525 IFK=1AND(A$="+"ORA$="-")THEN550
530 IFP=0ANDA$="."THENP=1:GOTO550
533 IFS=0ANDA$=CHR$(20)THEN537
534 IFS=1ANDA$=CHR$(20)THENS=0:GOTO510
535 IFS=1THEN545
536 S=1:GOTO510
537 IFB$="."THENP=0
540 PRINT"▌▌ ▌▌";:GOTO555
545 E=1:GOTO553
550 B$=A$:N$=N$+A$:PRINTA$;
551 NEXTK:PRINT:RETURN
553 K=L:NEXTK:PRINT:RETURN
555 N$=LEFT$(N$,LEN(N$)-1):GOTO510
```

All'inizio il programma chiede la lunghezza massima L del dato (nella quale devi calcolare anche il carattere RETURN), poi chiama il sottoprogramma di lettura in 500, che legge al massimo L caratteri. Il sottoprogramma ritorna il dato valido se l'indicatore E di errore risulta =0. Il dato rimane della sua lunghezza anche se <L. Il programma accetta un numero con segno e con un punto decimale; esso accetta

anche la cancellazione con il tasto DEL delle cifre numeriche e del punto decimale. Come vedi abbiamo dovuto controllare che si riceva un solo punto decimale. Osserva il controllo del tasto DEL (CHR$(20)) e del RETURN (CHR$13)) che viene recepito dall'istruzione GET, e l'uso di tre indicatori di errore: E, che viene controllato dal programma principale, P che serve per controllare la presenza di un solo punto decimale, e S che serve a cancellare il carattere precedente anche se era errato, cioè consente di non accettare il primo errore e di procedere correggendo. Ci sembra interessante esporre il diagramma a blocchi del programma INPUT1.

INPUT 1
START

QUANTI
CARATTERI
L

NO

BENE

SI

SCRIVI
DATO

SOTT.
LETTURA
500

DATO NON
VALIDO

NO

E = 0

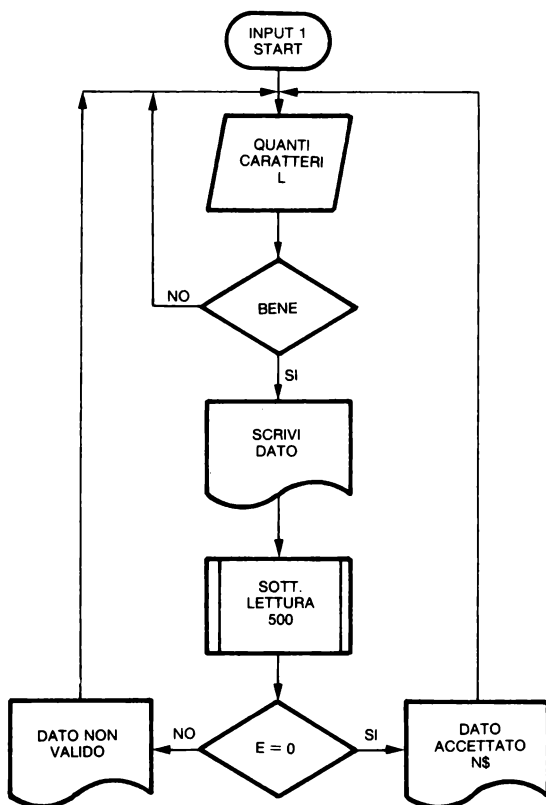
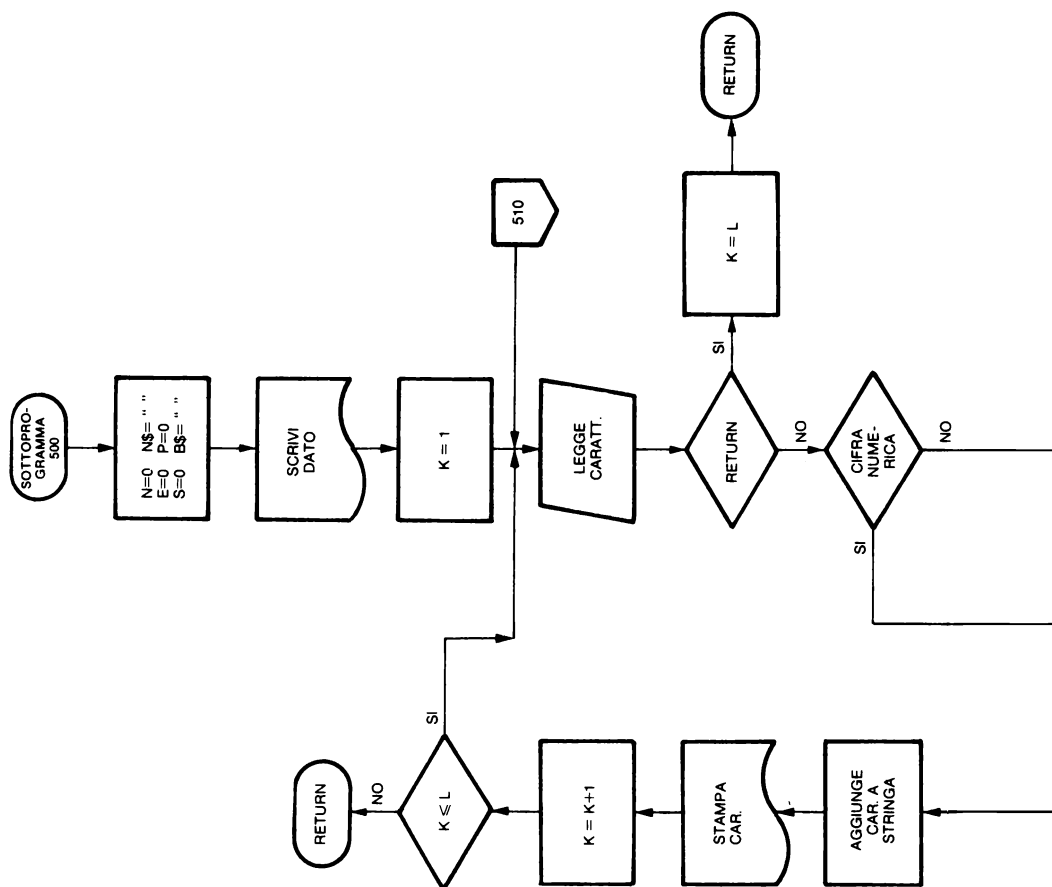
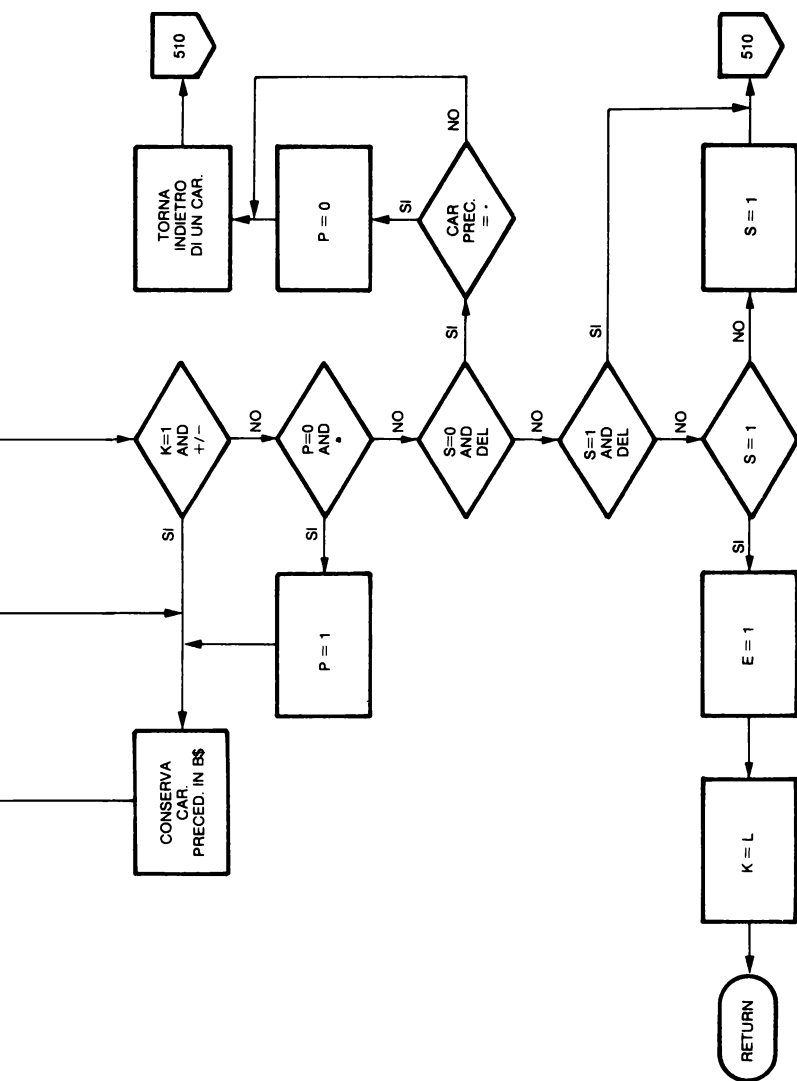
SI

DATO
ACCETTATO
N$

Figura 3.1 a) Diagramma a blocchi del programma INPUT1.

SOTTOPRO-
GRAMMA
500
N=0 N$=" "
E=0 P=0
S=0 B$=" "
SCRIVI
DATO
K = 1
510
LEGGE
CARATT.
RETURN
SI
K = L
RETURN
NO
CIFRA
NUME-
RICA
SI
NO
RETURN
NO
K ⩽ L
SI
K = K+1
STAMPA
CAR.
AGGIUNGE
CAR. A
STRINGA

Segue il programma INPUT2, nel quale si controlla un dato numerico letto come stringa con INPUT.

```
1 REM INPUT2
10 REM CONTROLLO DATO NUMERICO
15 REM LETTO CON INPUT
20 REM NOTA LA LUNGHEZZA MASSIMA DEL DATO
25 INPUT"]QQQUANTI CARATTERI: ";L
26 IFL<=0THENPRINT"]]":GOTO25
27 S$="":FORK=1TOL:S$=S$+" ":NEXTK
29 GOSUB500
30 IFE<>0THEN35
33 PRINT"SQQQQACCETTATO: "A$:GOTO40
35 PRINT"SQQQQNON VALIDO"
40 PRINT"PREMI UN TASTO PER CONTINUARE"
45 Z$="":GETZ$:IFZ$=""THEN45
50 GOTO25
500 PRINT"SQQQQQQQQQQQQQQQQQQQ"
505 P=0:E=0:A$="":INPUT"DATO: ";A$
515 IFLEN(A$)>LTHENA$=LEFT$(A$,L)
525 FORK=1TOLEN(A$):B$=MID$(A$,K,1)
530 IFK<>1THEN540
535 IFB$="+"ORB$="-"THEN555
540 IFB$>"/"ANDB$<":"THEN555
545 IFB$="."ANDP=0THENP=1:GOTO555
550 E=E+1:K=L:NEXTK:RETURN
555 NEXTK:A$=LEFT$(A$+S$,L):RETURN
```

All'inizio il programma chiede la lunghezza massima L del dato e ritorna un dato di L caratteri con eventuale troncamento o aggiunta di spazi. Se il dato non va bene il sottoprogramma ritorna l'indicatore E di errore $<>0$. Il dato viene letto nella parte bassa del video e riportato nella parte alta dopo l'accettazione.
Mediante l'indicatore P di errore per il punto controlliamo la presenza di un solo punto decimale. Nella fase di scrittura sotto INPUT siamo liberi di usare i tasti di controllo, tipo DEL o altri, che però sono gestiti dall'EDITOR e non fanno parte del dato ricevuto.

Segue il programma INPUT3, che controlla se un codice letto con INPUT in una variabile stringa ha uno dei 4 possibili valori.

```
1 REM INPUT3
10 DATAAC,SD,TU,XT
15 FORK=0TO3:READC$(K):NEXTK
20 INPUT"CODICE: ";A$
23 S=0:FORK=0TO3
27 IFA$=C$(K)THENS=1
29 NEXTK
30 IFS=0THEN20
35 PRINT"VA BENE":STOP
```

Seguendo i suggerimenti che puoi ricavare dagli esempi riportati, potrai adattarli alle situazioni che si prsenteranno nei tuoi programmi.

Naturalmente una buona routine di ingresso dati deve chiedere alla fine se i dati sono accettabili, presentandoli tutti sul video, con possibilità di andare a correggere quelli non buoni, pur essendo risultati validi da un punto di vista formale. Negli esempi del prossimo paragrafo presentiamo anche questa sequenza di programma.

3.5 PREPARAZIONE MASCHERE VIDEO

E' abbastanza raro il caso di un programma che non chieda dati di ingresso istituendo un colloquio video/tastiera. Il modo più semplice per chiedere e ricevere dati è una banalissima lista, preceduta dalla pulizia del video. Nella stesura dei programmi di questo libro ci siamo spesso limitati a questo; ora cerchiamo di fare anche qualcosa di meglio.

Cominciamo con un esempio di richiesta di data e di MENU' di scelta. La routine SCELTA che segue può essere usata in qualunque programma estendendone anche la portata.

```
1 REM SCELTA
5 CH$="":SZ$="                    "
7 CD$=""
10 GOSUB300:PRINT""SPC(8)"DATA ODIERNA: ";
13 PRINTG$;"/";M$;"/";A$
15 GOSUB350
20 IFR$="N"THEN10
100 PRINT"";:POKE53280,7:POKE53281,1
```

```
102 PRINTCH$;SZ$"     PROGRAMMA CBM 64"
103 PRINT
105 PRINT:PRINTSZ$" 1 - INIZIALIZZAZIONE"
110 PRINT:PRINTSZ$" 2 - AGGIORNAMENTO"
115 PRINT:PRINTSZ$" 3 - PROCEDURA CALCOLO"
120 PRINT:PRINTSZ$" 4 - STAMPA RISULTATI"
125 PRINT:PRINTSZ$" 0 - FINE"
130 PRINT:PRINT:PRINTSZ$"       OPZIONE: ";
135 GETT$:IFT$<"0"ORT$>"4"GOTO135
140 PRINT""+T$:GOSUB360
145 A=VAL(T$):IFA=0THEN9000
150 ONAGOTO1000,2000,3000,4000
155 STOP
300 POKE53280,5:POKE53281,2
303 PRINT"":OPEN1,0,1
305 PRINTSZ$"   DATA  GG,MM,AA";
310 INPUT#1,G$:PRINT"";
311 INPUT#1,M$:PRINT"";
312 INPUT#1,A$
313 DT$=G$+M$+A$:CLOSE1:RETURN
350 OPEN1,0,1
351 PRINTLEFT$(CD$,21);
353 PRINTSZ$"   CONFERMI ?  S";:INPUT#1,R$
355 IFR$<>"S"ANDR$<>"N"THEN351
357 CLOSE1:RETURN
360 A$="":PRINTCD$;
361 M$="PREMI UN TASTO PER PROSEGUIRE"
362 PRINT"     "M$
363 GETA$:IFA$=""THEN363
364 PRINTCH$:RETURN
1000 STOP
2000 STOP
3000 STOP
4000 STOP
9000 STOP
```

Il programma inizialmente prepara tre stringhe di costanti. La CD$ inizia con il carattere HOME e prosegue con 23 CRSR/giù; essa, se stampata tutta, porta alla linea 23 del video. Volendo ne puoi stampare una parte, usando la funzione LEFT$, e scendere alla riga desiderata.

Alla linea 10 chiamiamo il sottoprogramma 300 che richiede la data, poi la stampa e ne chiede conferma con il sottoprogramma in 350.
La routine di richiesta data usa le stringhe con il carattere CRSR/sinistra per posizionarsi su GG, MM, AA; tu devi scrivere il giorno e premere RETURN, scrivere il mese e premere RETURN e poi l'anno e premere RETURN. All'inizio essa modifica i colori del video; inoltre apre la tastiera come file per leggere (device=0) evitando il punto interrogativo di richiesta dati, così vedi solo il cursore che pulsa.
Poi il programma (da 100 a 155) propone un MENU' di scelta tra 5 opzioni diverse; osserva l'uso del campo inverso. La risposta 0 (zero) non può essere trattata con l'istruzione ON e quindi abbiamo posto un controllo alla linea 145.
I due sottoprogrammi in 350 e 360 sono di uso comune nei colloqui video/tastiera; il primo chiede conferma del quadro video e controlla la risposta S o N, il secondo crea un attesa fino alla pressione di un tasto per consentire di leggere il video.

Segue il programma QUADRO1, che è formato da due parti.
Il sottoprogramma che inizia in 107 serve per preparare una struttura di dati, con 15 campi aventi intestazioni e lunghezze diverse. E' chiaro che si tratta solo di un esempio inerente all'argomento che stiamo trattando; infatti non avrebbe senso strutturare i dati ogni volta che si usa il programma. Questo sottoprogramma dovrebbe far parte di un programma di inizializzazione che va poi a scrivere, su disco o su nastro, un file contenente le informazioni sulla struttura dei dati; tale file può poi essere riletto dai programmi e fornire i parametri necessari per gestire la struttura di dati creata.
Il sottoprogramma che inizia in 59 serve per leggere la sequenza dei dati e immagazzinarla nel vettore R$(15). Esso prevede la possibilità di correggere i dati introdotti. Nel sottoprogramma in 107 viene proposta una struttura di dati, con intestazione e lunghezza, e viene chiesta conferma o modifica per ogni campo, con la clausola che la lunghezza di ogni campo non superi 30 caratteri e che il totale delle lunghezze dei campi non superi 237. La lunghezza di un campo non comprende il carattere RETURN. La tastiera viene aperta come file per abolire il punto interrogativo in fase di Input; ricorriamo alla stringa CD$ e alla funzione TAB per posizionarci nei punti desiderati del video. Il vettore D$(15) contiene le descrizioni dei campi e il vettore L(15) le lunghezze. Aver disposto tutto in variabili con indice consente di programmare facilmente cicli.

```
1 REM QUADRO1
3 L=15:DIMD$(15),L(15),M$(11),M(11),R$(15)
5 CH$="J":CD$="SQQQQQQQQQQQQQQQQQQQQQQQ"
7 SZ$="        "
9 PRINT"J";:FORI=1TO30:SP$=SP$+" ":NEXTI
11 DATA8,6,9,0,5,5,6,8,0,4,9
13 FORK=1TO11:READM(K):NEXTK
```

```
15 M$(1)="XXXXSCELTA NOMI DEI 15 CAMPI"
17 M$(2)="CONFERMA IL NOME SE VA BENE"
19 M$(3)="ALTRIMENTI RISCRIVILO."
21 M$(4)=""
23 M$(5)="OSSERVA LE LUNGHEZZE DEI CAMPI"
25 M$(6)="PUOI CAMBIARLE. LA SOMMA DELLE"
27 M$(7)="LUNGHEZZE DEI CAMPI NON PUO´"
29 M$(8)="SUPERARE 237 CARATTERI,"
31 M$(9)=""
33 M$(10)="LA LUNGHEZZA DI UN CAMPO NON PUO´"
35 M$(11)="SUPERARE 30 CARATTERI."
37 MS$="          PREMI UN TASTO PER PROSEGUIRE■"
39 MT$="             QUALE CAMPO       ▌■■■▌"
41 PRINTCH$;:GOSUB107
43 GOSUB59:STOP
45 A$="":PRINTCD$;MS$
47 GETA$:IFA$=""THEN47
49 PRINTCH$:RETURN
51 CLOSE1:OPEN1,0,1:PRINTLEFT$(CD$,21);
53 PRINTSZ$"   CONFERMI     S▌▌";
55 INPUT#1,R$:IFR$<>"S"ANDR$<>"N"THEN51
57 CLOSE1:RETURN
59 CLOSE1:OPEN1,0,1
61 PRINT"   INGRESSO DATI"
63 F=0:PRINT" 1";TAB(4);D$(1)+":←▌▌";
65 INPUT#1,R$(1):PRINT
67 IFLEFT$(R$(1),1)="←"THENF=1:RETURN
69 FORI=2TOL:PRINTI;TAB(4);D$(I)":←▌▌";
71 INPUT#1,R$(I):PRINT:NEXTI:GOSUB85
73 GOSUB51:IFR$="S"THENCLOSE1:RETURN
75 CLOSE1:OPEN1,0,1
77 PRINTLEFT$(CD$,21);MT$;:INPUTR
79 IFR<1ORR>LTHEN77
81 IFR>LORR<1GOTO77
83 GOSUB91:GOTO73
85 FORI=1TOL
87 R$(I)=LEFT$(R$(I)+SP$,L(I))
89 NEXTI:RETURN
91 PRINT"";:FORI=1TOR-1
93 PRINTI;TAB(4);D$(I);":";R$(I):NEXTI
95 PRINTR;TAB(4);D$(R);":←▌▌";:INPUT#1,R$(R)
97 PRINT:R$(R)=LEFT$(R$(R)+SP$,L(R))
```

```
99 IFR>=LTHEN105
101 FORI=R+1TOL
103 PRINTI;TAB(4);D$(I);":";R$(I):NEXTI
105 RETURN
107 DW$="S":D$(1)="COGNOME":D$(2)="NOME"
109 D$(3)="INDIRIZZO":D$(4)="TELEFONO"
111 D$(5)="C.A.P.":D$(6)="CITTA'"
113 FORIK=7TO15:D$(IK)="C"+CHR$(42+IK):NEXTIK
115 DATA20,15,30,15,8,15,22,22,15,15
117 DATA15,15,15,10,5
119 FORIK=1TO15:READL(IK):NEXTIK
121 CLOSE1:OPEN1,0,1:PRINTCH$;
123 FORK=1TO11
125 PRINTSPC(M(K))"▣"M$(K)"▤":NEXTK
127 GOSUB45
129 PRINTCH$"▣CONFERMA STRUTTURA DATI▤"
131 FORJ=1TO15
133 PRINT"L=";L(J);TAB(10);D$(J):NEXTJ
135 PRINT:PRINT"SE VA BENE PREMI RETURN"
137 PRINT"SE NO RISCRIVI PER SOSTITUIRE"
139 FORJ=1TO15
141 PRINTLEFT$(CD$,J+1);TAB(10);"▣";D$(J);"▤";
143 PRINTCD$;SP$;CD$;:T$="":INPUT#1,T$
145 IFT$=""THEN149
147 D$(J)=T$
149 PRINTLEFT$(CD$,J+1);TAB(8);SP$;
151 PRINTLEFT$(CD$,J+1);TAB(10);D$(J)
153 PRINTLEFT$(CD$,J+1);TAB(2);"▣";L(J);"▤";
155 PRINTCD$;SP$;CD$;:IL=0:INPUT#1,IL
157 IFIL=0THEN165
159 IFIL>30THEN153
161 L(J)=IL
163 PRINTLEFT$(CD$,J+1);TAB(4);"  ";
165 PRINTLEFT$(CD$,J+1);TAB(2);L(J):NEXTJ
167 YX=0:FORJ=1TO15:YX=YX+L(J):NEXTJ
169 IFYX<=237THEN177
171 PRINTLEFT$(CD$,20);
173 PRINT"TROPPI CARATTERI";YX
175 GOSUB45:GOTO129
177 GOSUB51
179 IFR$="N"THEN121
181 CLOSE1:RETURN
```

Nel sottoprogramma in 59 si apre ancora la tastiera come file e si leggono i dati con INPUT # tutti come stringhe. Non abbiamo svolto un controllo dei campi, ma potrebbe essere introdotto per particolari valori degli indici, per controllare, come abbiamo già visto nel precedente paragrafo, i campi numerici o altro. Il sottoprogramma consente la modifica dei dati introdotti. Per uscire dal sottoprogramma, nel caso faccia parte di un programma che legge dati ciclicamente, abbiamo posto il controllo sul primo campo; se si risponde con RETURN resta valido il carattere "freccia sinistra" e viene riconosciuto.
Il sottoprogramma in 45 serve per creare un ciclo di attesa fino alla pressione di un tasto. Quello in 51 chiede conferma del quadro video presente.

Segue il programma QUADRO2, nel quale abbiamo cercato di realizzare un quadro video più bello, giocando sui colori e sui riquadri in campo inverso.

```
1 REM QUADRO2
4 V$="■■■■■■■■■■■■■■■■■■■■■■■■■■"
7 SP$="■                              "
10 DIMD$(12),R$(12),CR(12),CC(12),L(12)
13 D$(1)="■1)COGNOME■":D$(2)="■2)NOME■"
16 D$(3)="■3)INDIRIZZO■":D$(4)="■4)CAP■"
19 D$(5)="■5)CITTA'■":D$(6)="■6)TELEFONO■"
22 D$(7)="■7)LUOGO NASCITA■"
25 D$(8)="■8)DATA NASCITA■"
28 D$(9)="■9)COD. FISCALE■"
31 D$(10)="■10)PART. IVA■"
34 D$(11)="■11)PROFESSIONE■"
37 D$(12)="■12)STATO CIVILE■"
40 T$="■INGRESSO  DATI■"
43 DATA2,2,5,8,8,11,14,14,17,17,20,20
46 DATA2,25,4,2,15,10,2,24,4,24,2,24
49 DATA20,15,30,8,15,15,15,8,15,15,20,8
52 FORK=1TO12:READCR(K):NEXTK
55 FORK=1TO12:READCC(K):NEXTK
58 FORK=1TO12:READL(K):NEXTK
61 POKE53280,7:POKE53281,1
64 PRINT"■"TAB(12)T$
67 FORK=1TO12
70 PRINTLEFT$(V$,CR(K)+1)TAB(CC(K));D$(K)
73 POKE646,7
76 PRINTLEFT$(V$,CR(K)+2);
77 PRINTTAB(CC(K))LEFT$(SP$,L(K)+1)
79 POKE646,14
```

```
82 NEXTK
85 POKE646,2:FORK=1TO12
88 PRINTLEFT$(V$,CR(K)+2)TAB(CC(K));"■ ■■■";
91 GOSUB154:FORI=1TO200:NEXTI
94 NEXTK
97 POKE646,6
100 POKE198,0
103 PRINTV$"                                    ";:R$=""
106 PRINTV$"              CONFERMI (S/N)";:INPUTR$
109 IFR$<>"S"ANDR$<>"N"THEN103
112 IFR$="S"THEN139
115 POKE198,0
118 PRINTV$"                                    ";:R$=""
121 PRINTV$"               QUALE CAMPO:";:INPUTR$
124 K=VAL(R$):IFK<1ORK>12THEN115
127 POKE646,2
130 PRINTLEFT$(V$,CR(K)+2)TAB(CC(K));"■ ■■■";
133 GOSUB154
136 GOTO97
139 POKE646,14:PRINT"□         DATI LETTI"
142 FORK=1TO12
145 PRINTD$(K);":  ";R$(K)
148 NEXTK
151 STOP
154 POKE198,0:I=1:R$(K)=""
157 A$="":GETA$:IFA$=""THEN157
160 IFA$=CHR$(13)THEN175
163 PRINTA$;
166 R$(K)=R$(K)+A$
169 IFI=L(K)THEN175
172 I=I+1:GOTO157
175 POKE198,0:FORI=1TO200:NEXTI:RETURN
```

Anche qui ricorriamo ai vettori per memorizzare intestazioni (D$(12)), lunghezze (L(12)) e valori (R$(12)) dei dati. In più abbiamo creato un vettore CR(12), per contenere il numero della riga video dove vogliamo posizionare il dato, e un vettore CC(12) per contenere il numero da usare per la posizione sulla riga. Questi due vettori servono, per la funzione LEFT$ che preleva la parte necessaria della stringa V$ e manda per mezzo dell'istruzione PRINT alla riga desiderata, e per la funzione TAB che crea la posizione sulla riga. Il quadro video viene preparato sfruttando il cambio dei colori e il campo inverso. Dopo aver posto sul video le descrizioni e la

numerazione di ogni campo, creiamo nella riga sottostante un rettangolo in campo inverso per contenere il dato, esattamente della lunghezza del dato da ricevere. Quest'ultimo accorgimento aiuta a non scrivere dati troppo lunghi. I dati sono letti con GET carattere per caratere dal sottoprogramma in 154. Non è stato necessario aprire la tastiera come file, dato che GET non crea il punto interrogativo.
Anche qui i dati possono essere corretti scegliendo il campo da correggere.
La lunghezza dei campi può essere quella stabilita o può essere minore premendo il RETURN. Il programma non accetta campi più lunghi dei limiti contenuti in L(K).
Osserva l'uso di POKE 198,0 per svuotare il buffer della tastiera, e dei cicli d'attesa con FOR per rallentare le operazioni.
Il sottoprogramma in 154 non fa controlli sui caratteri introdotti, salvo per la lungheza dei campi e il RETURN. Devi stare attento all'effetto che può provocare l'uso del tasto DEL, che viene accettato come un qualunque carattere, produce sul video l'effetto di cancellazione, ma influisce sul computo del numero dei caratteri.
La routine in 154 può essere modificata, nel modo mostrato nel precedente paragrafo, o in altro, per fare accettare solo determinati gruppi di caratteri.

CAPITOLO 4

MEMORIZZAZIONE E CARICAMENTO PROGRAMMI

4.1 INTRODUZIONE

In questo Capitolo ci occupiamo della gestione dei programmi relativamente ai supporti di memorizzazione cassetta e disco e all'utilizzo della memoria.

Il linguaggio Basic dispone dei comandi LOAD, SAVE e VERIFY, che consentono di caricare un programma in memoria, di memorizzarlo sul supporto e di verificare se la registrazione è buona.
Per eseguire questi comandi non è necessario stabilire una comunicazione con la periferica per mezzo del comando OPEN e quindi non serve neanche il comando CLOSE.
Per quanto riguarda il disco vedremo anche alcuni comandi particolari che consentono operazioni di servizio sui file di programmi.
Rimandiamo al volume che seguirà subito questo la trattazione di tutta la problematica relativa al trattamento dei file di dati su disco e cassetta.
Abbiamo pensato di farti cosa gradita aggiungendo i Paragrafi 4.5 e 4.6, ma te ne raccomandiamo la lettura solo se sei già diventato un po' esperto di programmazione.

Ti consigliamo, se desideri sfruttare in pieno le possibilità del tuo calcolatore, di acquistare uno di quei programmi di utilità (tipo TOOL-KIT, o simili) che consentono di:
...numerare automaticamente le linee del programma mentre si scrive;
...rinumerare un programma che abbia subito molti aggiustamenti;
...cancellare blocchi di programma;
...ricercare una particolare stringa di caratteri nel programma;
...sostituire automaticamente una stringa di caratteri con un'altra;
...assegnare funzioni utili ai tasti funzione posti sulla destra della tastiera;
...ricercare errori;
...visualizzare i numeri delle linee di programma che si eseguono;
...fondere programmi diversi in un unico programma.

Abbiamo trattato la struttura dei programmi nella memoria del calcolatore nel Paragrafo 8.3; qui ci basta ricordare che ogni linea di programma contiene due byte, chiamati LINK, che puntano all'indirizzo della prossima istruzione. I programmi Basic, in condizioni normali, sono situati in memoria a partire dal byte di indirizzo 2049. I valori di LINK nascono in conseguenza. Noi possiamo spostare il puntatore all'inizio del programma Basic, agendo sui byte 43 e 44 della pagina 0 della RAM. Se carichiamo un programma, dopo aver operato questo spostamento, esso viene RILOCATO, cioè viene caricato a partire dal nuovo indirizzo e i due byte di LINK vengono modificati.
Nei comandi SAVE e LOAD si può aggiungere un parametro, come SA, dopo il DN della periferica, per intervenire sulla rilocazione. L'effetto è il seguente:
..SAVE con SA=1 memorizza il programma in modo tale che in fase di caricamento con un LOAD senza SA il programma non viene rilocato;
..LOAD con SA=1 carica il programma salvato senza SA senza rilocarlo.
Devi usare con cautela queste opzioni; infatti dopo aver caricato un programma che non inizia nel modo solito, dovrai preoccuparti di modificare il puntatore (byte 43 e 44) per poterlo far partire.

Nel seguito i parametri compresi nella spiegazione delle istruzioni possono comparire sia in lettere minuscole che maiuscole.

4.2 FILE DI PROGRAMMI SU CASSETTA

Per memorizzare un programma presente in memoria su cassetta possiamo scrivere:

1) SAVE

2) SAVE ""

3) SAVE "nome"

4) SAVE "nome",1

5) SAVE "nome",1,1

6) SAVE "nome",1,2

7) SAVE "nome",1,3

che sono solo alcuni casi particolari del formato:

SAVE [fn] [,dn] [,sa]

dove:
..fn è il nome del file e può mancare, se presente vengono accettati solo i primi 16

caratteri;
..dn è il numero della periferica, che per la cassetta è 1, e può essere omesso;
..sa è un parametro che influisce sulla rilocazione dei programmi se è posto a 1, come già abbiamo visto nel precedente paragrafo; per la cassetta esso può assumere anche il valore 2, che significa scrittura con segnalazione finale di fine nastro (EOT), e il valore 3 che significa sia non rilocabile che con EOT finale.

Si possono memorizzare i programmi senza nome, ma a nostro avviso, non è consigliabile. E' sempre meglio assegnare un nome al programma e riportarlo sulla etichetta esterna della cassetta.

Ti raccomandiamo di mantenere il registratore sempre in stato di riposo, senza tasti abbassati, questo per evitare che parta senza aver prima sistemato la cassetta. Se vuoi controllare dove arriva la registrazione, devi azzerare il contagiri. Inoltre è meglio riavvolgere la parte iniziale del nastro non utlizzabile prima di inserire la cassetta nel registratore.

Quando premi il tasto RETURN per fare accettare il comando, o quando esso va in esecuzione da programma, compare sul video il messaggio:

PRESS RECORD & PLAY ON TAPE

a quel punto devi premere PRIMA il tasto RECORD e POI il tasto PLAY, assolutamente no il viceversa. Infatti la registrazione inizia quando si preme PLAY, e, se il tasto RECORD non è premuto, si rischia di perderne una parte.
Durante la registrazione scompaiono le scritte dal video e riappaiono quando essa è terminata. Il sistema registra due volte i blocchi di programma; in fase di lettura controlla che non ci siano errori e differenze tra i due blocchi.

Dopo aver memorizzato un programma su nastro è bene farne la verifica. Puoi scrivere:

VERIFY "nome",1 o anche solo VERIFY

naturalmente devi riavvolgere il nastro al punto giusto. Per evitare confusioni è bene usare il nome che si è assegnato al programma; infatti, se non hai riavvolto bene il nastro e sei andato troppo indietro, viene ricercato il programma, da confrontare con quello in memoria, mediante il nome.
Dopo il comando compare la richiesta:

PRESS PLAY ON TAPE

anche in questo caso quando hai accontentato la richiesta, scompaiono le scritte dal video, per ricomparire alla fine con il messaggio di OK o di errore.

Il programma viene registrato su nastro esattamente come si trova in memoria, byte dopo byte.

Per caricare in memoria un programma da cassetta si può scrivere:

1) LOAD ""

2) LOAD "nome"

3) LOAD "nome",1

4) LOAD "nome",1,1

che sono solo alcuni casi particolari del formato:

LOAD [fn] [,1] [,1]

dove i parametri hanno il significato già visto.

Se manca il nome viene caricato il primo programma che viene incontrato sul nastro.

Dopo l'accettazione del comando compare la richiesta:

PRESS PLAY ON TAPE

e quando esegui scompaiono le scritte dal video; dopo un pò ricompaiono le scritte e c'è la segnalazione:

FOUND...

se qualcosa è stato trovato. Dopo scompaiono nuovamente le scritte e viene caricato il programma. Al termine ricompaiono le scritte con segnalazione LOADING/READY se è andato tutto bene, oppure di errore.

4.3 COMANDI PER LA GESTIONE DEL DISCO

L'unità 1541 collegata al COMMODORE 64 consente di usare i floppy disk come supporti di memorizzazione.
I floppy disk sono un buon supporto di memorizzazione ed arricchiscono le possibilità del calcolatore.
Per ora qui ci limitiamo a fornirti le informazioni necessarie per poter memorizzare programmi su disco e per poterli ricaricare in memoria.

I dischetti devono essere maneggiati con cura, senza piegarli e senza toccare le parti visibili dalle aperture del contenitore. Quando si acquista un dischetto nuovo, esso non può esssere adoperato se non si esegue su di esso l'operazione di FORMATTAZIONE, che vedremo tra poco.

Sull'unità, a sinistra, è presente un indicatore luminoso; esso diventa verde quando si dà corrente. Quando è in esecuzione un'operazione disco, l'indicatore diventa rosso, e, al termine torna verde. Se vedi pulsare in rosso l'indicatore significa che l'operazione in corso ha difficoltà ad essere eseguita. In certi casi compare anche un messaggio di errore dopo molti tentativi di esecuzione. Conviene spegnere l'unità, aspettare un breve intervallo di tempo e poi riprovare; se le condizioni di errore persistono può esserci un guasto nell'unità, ma può anche essere rovinato il dischetto.

All'interno dell'unità 1541 è presente un microprocessore con 16K di memoria ROM e 2K di memoria RAM. La memoria ROM contiene il Sistema Operativo del disco (DOS - Disk Operating System) che provvede all'esecuzione delle operazioni che vengono lanciate dal calcolatore verso la periferica. La memoria RAM serve per i buffer necessari a contenere le informazioni che vanno scritte sul floppy o che sono lette da esso, e l'area di lavoro del DOS.

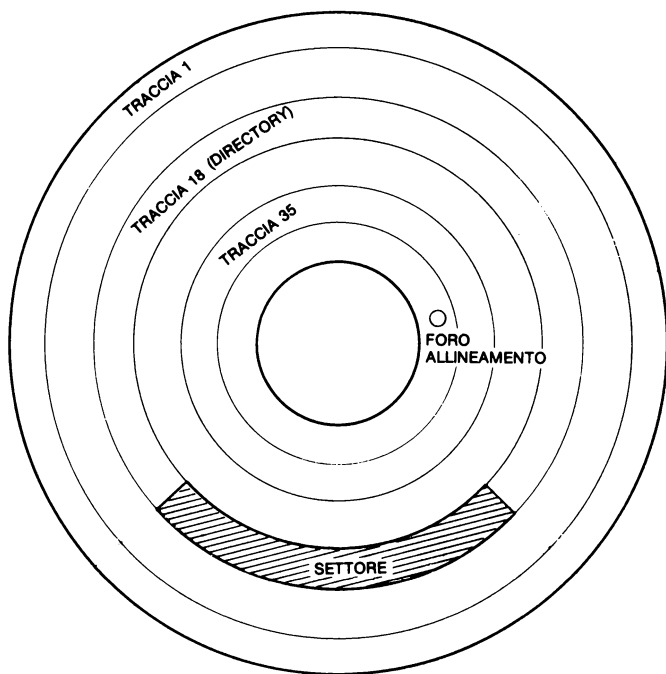



Figura 4.1 Schema di un floppy disk

Nella figura 4.1 è riportato uno schema semplificato del dischetto. Puoi vedere una parte tratteggiata con la dicitura SETTORE; questa è la parte di dischetto che viene coinvolta in una operazione di lettura o scrittura. In un SETTORE o BLOCCO sono contenuti 256 caratteri (256 byte) disponibili per l'utente.
Nella figura sono segnate le tracce; 1, 35 e 18; di queste tracce concentriche sul floppy ne abbiamo disponibili 35. Nella figura è segnata la prima e l'ultima; la numero 18 è accompagnata dal nome "directory", infatti essa è una traccia un pò speciale e serve per registrare l'indice del contenuto del disco (un pò come l'indice di un libro).
Il numero di settori disponibili su ogni traccia non è sempre uguale, varia per gruppi di tracce. La ripartizione è la seguente:

Num. traccia	Quanti settori	Num. settori
da 1 a 17	21	da 0 a 20
da 18 a 24	19	da 0 a 18
da 25 a 30	18	da 0 a 17
da 31 a 35	17	da 0 a 16

Il numero totale di settori disponibili risulta:

21x17+7x19+6x18+5x17=683

ma di questi, i 19 della traccia 18 sono riservati al sistema, e quindi restano disponibili:

683-19=664 settori.

Ogni settore contiene 256 byte; in totale un floppy disk mette a disposizione:

664x256=169984 byte.

Per poter usare il dischetto su di esso devono essere registrate alcune indicazioni; esse sono gli indirizzi delle tracce e dei settori e le informazioni iniziali della DIRECTORY. L'operazione che fa queste registrazioni iniziali si chiama FORMATTAZIONE. Si può formattare un disco nuovo e si ottiene un disco pronto per l'uso. Si può anche formattare un disco già usato, si ottiene un disco pronto per l'uso che ha perso tutte le informazioni che conteneva precedentemente.

Il FORO DI ALLINEAMENTO che compare nella figura serve per allineare il dischetto rispetto alla testina di lettura e scritura dell'unità, al momento dell'introduzione. Questo allineamento avviene automaticamente quando si introduce il dischetto nell'unità, ma può anche essere ottenuto con un comando da programma. Il comando prende il nome di INIZIALIZZAZIONE.

Il DOS esegue le operazioni di lettura e scrittura relative al disco e gestisce l'indice del disco. In sostanza esso gestisce la comunicazione tra il disco e l'interfaccia seriale che lo collega al calcolatore.

Per stabilire una comunicazione con il disco il calcolatore può fare le seguenti cose:

.. inviare un comando di LOAD;
.. inviare un comando di SAVE;
.. inviare un comando di OPEN.

Delle operazioni di LOAD e SAVE ci occupiamo nel prossimo paragrafo; vediamo qui cosa succede con OPEN.

Il formato della OPEN è:

    OPEN lfn,dn,sa

dove:

lfn, deve essere il numero logico del file che si vuole usare per comunicare;

dn, è il numero della periferica, e, nel caso di una sola unità collegata, risulta 8;

sa, indirizzo secondario, prende in questo caso il nome di CANALE; esso può variare da 2 a 15.

In realtà per la periferica sono disponibili i numeri di canale da 0 a 15, ma 0 e 1 sono SA usati dalle istruzioni LOAD e SAVE, quindi la OPEN può usare i numeri da 2 a 15. Precisamente i numeri da 2 a 14 per le operazioni relative ai file di dati, che non trattiamo qui, il 15 per la trasmissione dei comandi.

Prendiamo in considerazione la:

OPEN N,8,15 usata per impartire comandi al disco. N può essere scelto a piacere, noi, per abitudine, usiamo il numero 15.

Naturalmente dopo aver impartito comandi come vedremo a seguito di questa OPEN, è necessario usare il comando CLOSE per chiudere la comunicazione; esso si scrive:

    CLOSE N

usando lo stesso lfn usato nella OPEN.

A seguito della OPEN impartiamo i comandi con una stringa nella istruzione:

PRINT# lfn,stringa-comandi

I comandi che possono essere impartiti sono i seguenti:

.. NEW formattazione dischetto;
.. INITIALIZE inizializzazione dischetto;
.. VALIDATE controllo e sistemazione dischetto;
.. COPY copia di un file;
.. RENAME cambio nome di un file;
.. SCRATCH cancellazione di un file.

Per indicare un comando è sufficiente indicare la prima lettera.

Esaminiamo ora dettagliatamente ogni comando. Sono possibili due procedure:
.. usare la stringa comando con una istruzione PRINT# ;
.. usare la stringa comando direttamente nella OPEN, facendola seguire dopo gli altri parametri.

Comando NEW

Prepara un disco nuovo per l'uso o cancella un disco già usato e lo rende nuovamente disponibile. Sequenze possibili:

OPEN15,8,15:PRINT# 15,"N0:FN,XX"

OPEN15,8,15,"N0:FN,XX"

FN è il nome da assegnare al disco e può essere lungo al massimo 16 caratteri. XX è un codice a due caratteri di identificazione del disco; esso non può essere omesso se si sta formattando un disco nuovo. La presenza dei due caratteri di identificazione fa sì che vengano scritti sul disco gli indirizzi di traccia e settore. Se si formatta un disco già usato e si omette l'identificazione, viene mantenuta la precedente identificazione, può essere cambiato il nome e il disco viene cancellato impiegando un tempo minore; infatti non vengono riscritti tutti gli indirizzi di traccia e settore. Lo zero dopo N identifica il disco; se si usasse un'altra periferica con due dischi, si avrebbe disco 0 e disco 1.

Comando INITIALIZE

Serve per caricare nella RAM dell'unità le informazioni necessarie per gestire il

disco e ad allineare fisicamente il dischetto rispetto alla testina di lettura e scrittura.
Sequenze possibili:

OPEN15,8,15:PRINT# 15,"I"

OPEN15,8,15,"I"

dove a I si può volendo far seguire 0, come numero del disco, ma non è necessario avendo una unità disco semplice (non con due dischetti).

Per una buona gestione del disco, esso deve essere inizializzato. Il sistema lo fa automaticamente al momento della introduzione del dischetto, se osservi l'unità vedi accendersi la luce rossa per un brevissimo tempo e senti girare il dischetto. Ti consigliamo di inserire nei tuoi programmi, soprattutto se prevedono l'operazione di cambio dei dischetti, il comando di inizializzazione.

E' importante non confondere i due comandi di formattazione e inizializzazione; il primo scrive materialmente qualcosa sul dischetto, il secondo scrive qualcosa nella RAM e fa girare il dischetto per sistemarlo nell'alloggiamento.
I dischetti si pongono in rotazione solo quando svolgono operazioni di lettura o scrittura, altrimenti sono in stato di quiete.

Comando VALIDATE

Serve per mettere ordine in un disco dove siano presenti delle situazioni irregolari, tipo file non chiusi. Il comando sistema le zone libere o occupate sul dischetto leggendo il contenuto dell'indice (Directory). Non si può usare se il disco ha registrazioni non riportate nella Directory. Sequenze possibili:

OPEN15,8,15:PRINT# 15,"V"

OPEN15,8,15,"V"

dove si può volendo far seguire a V il numero 0 per il disco.

Un disco non è sicuramente in buone condizioni se la somma dei settori liberi e occupati rilevabile dalla lista della Directory non dà 664.

Comando COPY

Consente di copiare un file assegnandogli un altro nome (un programma memorizzato due volte con nomi diversi). Sequenze possibili:

OPEN15,8,15:PRINT# 15,"C:DFN=SFN"

OPEN15,8,15,"C:DFN=SFN"

dove:

DFN, è il nome del file destinazione (la copia che si vuole avere);
SFN, è il nome del file sorgente (quello che si vuole ricopiare).
I nomi possono essere al massimo di 16 caratteri.

Comando RENAME

Consente di cambiare il nome ad un file; viene cambiato il nome nella Directory senza spostare il file da dove si trova sul dischetto. Sequenze possibili:

OPEN15,8,15:PRINT# 15,"R:NFN=OFN"

OPEN15,8,15,"R:NFN=OFN"

dove:
NFN, è il nuovo nome che si vuole asssegnare al file;
OFN, è il vecchio nome del file.

Anche qui si può usare dopo R il numero 0 per il disco.

Comando SCRATCH

Serve per cancellare file dal dischetto. Sequenze possibili:

OPEN15,8,15:PRINT# 15,"S:FN"

OPEN15,8,15,"S:FN"

Per leggere nella memoria del calcolatore la Directory del disco si usa il comando:

LOAD"$",8

e per vederne il contenuto si scrive:

LIST.

Per listare la Directory sulla stampante si scrive:

OPEN4,4:CMD4:LIST

e poi, dopo la lista:

PRINT# 4:CLOSE4.

Naturalmente dopo avere usato i comandi descritti resta aperta la comunicazione tramite il canale 15; per chiuderla devi eseguire: CLOSE15.

Quando si usano comandi che richiedono nomi di file FN, si possono usare le seguenti semplificazioni:

* per rimpiazzare la parte finale di un nome, non importa di quanti caratteri;

? per rimpiazzare un carattere anche all'interno di un nome.

PRINT#15,"S0:PIP*" cancella tutti i file che hanno il nome che inizia con PIP.

PRINT#15,"S0:AE??GH" cancella tutti i file che hanno un nome di 6 caratteri, che inizia con AE e termina con GH.

4.4 FILE DI PROGRAMMI SU DISCO

Per la gestione dei programmi sul disco sono disponibili gli stessi comandi che per la cassetta, solo che in questo caso è obbligatorio citare il numero della periferica, DN. Un'altra differenza è che non viene chiesto di avviare l'unità; se non è stato inserito il dischetto si vede pulsare la luce di errore.
I comandi vanno dati dopo che il dischetto è stato inizializzato con il comando I, soprattutto se esso è stato appena cambiato.

Per memorizzare un programma su disco si scrive:

SAVE"0:FN",8

dove:
0, può essere omesso (numero disco);
FN, è il nome da assegnare al file programma;
8, è il numero della periferica.

Se fai seguire a 8 la virgola e il numero 1 (...,8,1), ottieni di salvare il programma in

modo che al momento del LOAD non venga rilocato.

Si può scrivere il comando come segue:

SAVE"@0:FN",8

per ottenere che il file, eventualmente già presente con lo stesso nome, venga cancellato e sostituito da quello nuovo.
Noi preferiamo seguire la procedura di cancellare prima con il comando SCRATCH il vecchio file, e poi usare il comando SAVE senza il carattere @.

Dopo il SAVE è raccomandabile l'uso del comando di verifica, che si scrive:

VERıFY"0:FN",8 oppure

VERIFY"*",8

e produce l'effetto di confrontare il programma di nome FN, oppure l'ultimo memorizzato, con quello presente in memoria.Se la verifica non dà buon esito si deve cancellare la registrazione e ripetere la procedura di SAVE dopo aver nuovamente inizializzato il dischetto.

Per caricare un programma da disco si scrive:

LOAD"0:FN",8

Come sempre il numero 0 per il disco può essere omesso.
Se il programma non esiste sul dischetto si vede pulsare la luce rossa; in questo caso devi inizializzare nuovamente il dischetto, leggere la Directory per controllare i nomi dei programmi presenti ed eventualmente cambiare dischetto.
Se il caricamento è terminato senza segnalazione di errore puoi far partire il programa o con RUN o con GOTON.
L'operazione di LOAD chiude tutti i canali di comunicazione tra calcolatore e disco; dovrà provvedere il programma a aprire i canali necessari con OPEN.
Usando dopo 8 lo SA uguale a 1 ottieni di non rilocare il programma che carichi.

4.5 OVERLAY DI PROGRAMMI

Quando si carica un programma usando il comando LOAD da tastiera si opera praticamente un NEW del calcolatore, cioè si azzera tutto prima di caricare il programma. Inoltre il programma parte a comando da tastiera con RUN o con GOTO.

Quando, invece, il comando LOAD viene eseguito da un programma non si ha l'azzeramento e il programma parte automaticamente, per effetto del nuovo caricamento il vecchio programma viene cancellato.
Quando si programma una procedura complessa che richiede parecchi programmi, è comodo poter alternare in memoria i programmi senza perdere le variabili, ma facendo in modo che un programma ritrovi le variabili lasciate dall'altro. Per ottenere questo bisogna far iniziare le variabili sempre dallo stesso indirizzo di memoria indipendentemnte dalla lunghezza dei programmi.
L'alternanza dei programmi in memoria prende il nome di OVERLAY.
Per risolvere il problema nel migliore dei modi si deve considerare il programma più lungo e rilevare il valore dei puntatori all'inizio delle variabili, aumentare di poco per lasciare un pò di margine per eventuali modifiche del programma e prendere nota dei valori trovati.
Si deve poi creare un programma di lancio della procedura; in esso con delle istruzioni POKE si modificano i tre puntatori di inizio variabili, inizio array e fine array, che partono inizialmente dagli stessi valori. L'ideale sarebbe anche inizializzzare nel programma di lancio tutte le variabili che usa la procedura, in modo da fissarle in memoria, salvo i corpi delle stringhe, naturalmente; questo fa guadagnare velocità ai programmi. Il programma di lancio esegue il LOAD del primo programma della procedura, questo poi esegue il LOAD del secondo e così via; le variabili non vengono toccate e tutti i programmi le possono usare.
Si devono usare degli accorgimenti, per esempio non possono mettersi in comune stringhe definite all'interno delle linee di un programma; infatti per stringhe di questo tipo la testata si trova tra le altre variabili, ma il puntatore manda all'interno del programma e quando se ne carica un altro, a quell'indirizzo si trovano altre cose. Le variabili con indice devono essere dimensionate nel programma di lancio.

Riportiamo un semplice esempio; tre programmi: LANCIO, CORTOMOD e LUNGOMOD. Il programma LANCIO sposta i tre puntatori molto avanti, anche se i tre programmi esempio sono corti, e carica CORTOMOD. Questo chiede una stringa di Input, poi stampa le sue variabili alla stampante e carica LUNGOMOD. Questo stampa le variabili ereditate dal programma precedente e poi le sue.

```
1 REM LANCIO CORTOMOD
5 POKE45,0:POKE46,16
6 POKE47,0:POKE48,16
7 POKE49,0:POKE50,16
8 PRINT"CARICO PROGRAMMI"
10 LOAD"CORTOMOD",8
15 END
```

```
1 REM CORTOMOD
2 C$="SCRIVI CC$:"
3 PRINTC$;:INPUTCC$
5 OPEN4,4:CMD4
6 PRINT"RUN CORTOMOD"
10 A=123
15 B=678
25 PRINT"A=";A,"B=";B
26 PRINT"CC$=";CC$
27 PRINT#4:CLOSE4
30 LOAD"LUNGOMOD",8
```

```
1 REM LUNGOMOD
2 REM PROVA CONSERVAZIONE VARIABILI
3 REM A,B,C$ DEVONO MANTENERSI
5 OPEN4,4:CMD4
6 PRINT"RUN LUNGOMOD"
10 D=998
15 E=790
20 F$="LUNGOMOD"
25 PRINT"A=";A,"B=";B
26 PRINT"CC$=";CC$
28 PRINT"D=";D,"E=";E
29 PRINT"F$=";F$
30 PRINT#4:CLOSE4
40 END
```

RISULTATI PROGRAMMI PRECEDENTI

```
RUN CORTOMOD
A= 123              B= 678
CC$=CORTOMOD
```

```
RUN LUNGOMOD
A= 123              B= 678
CC$=CORTOMOD
D= 998              E= 790
F$=LUNGOMOD
```

Dopo aver provato questi programmi devi ripristinare i valori dei puntatori con delle istruzioni POKE o spegnere il calcolatore per cancellare la memoria.

Per fare una controprova riportiamo i due programmi CORTO e LUNGO, che non modificanno i puntatori, dai risultati puoi vedere che il secondo programma, più lungo del primo, invade la zona variabili del primo e quindi non esistono più le prime variabili.

```
1 REM CORTO
5 OPEN4,4:CMD4:PRINT"RUN CORTO"
10 A=123
15 B=678
20 C$="CORTO"
25 PRINT"A=";A,"B=";B
26 PRINT"C$=";C$
27 PRINT#4:CLOSE4
30 LOAD"LUNGO",8
```

```
1 REM LUNGO
5 OPEN4,4:CMD4:PRINT"RUN LUNGO"
10 D=998
15 E=790
20 F$="LUNGO"
25 PRINT"A=";A,"B=";B
26 PRINT"C$=";C$
28 PRINT"D=";D,"E=";E
29 PRINT"F$=";F$
30 PRINT#4:CLOSE4
40 STOP
50 END
```

RISULTATI PROGRAMMI PRECEDENTI

```
RUN CORTO
A= 123              B= 678
C$=CORTO


RUN LUNGO
A= 0            B= 0
C$=
D= 998              E= 790
F$=LUNGO
```

4.6 PROGRAMMI E SUGGERIMENTI UTILI

Prima di abbandonare l'argomento toccato nel precedente paragrafo, desideriamo suggerirti un metodo da adottare quando si provano programmi complessi, nei quali sono probabilmente contenuti errori.
Questa implementazione del Basic quando si interrompe un programma e si fanno delle modifiche o si entra in EDITOR, cancella tutte le variabili già in memoria. Per evitare questo fatto, che a volte risulta fastidioso, basta, dopo aver caricato il programma, leggere in modo immediato il contenuto dei byte 45 e 46 (puntatori inizio variabili), aggiungergli qualcosa (per un migliaio di byte basta aggiungere 4 al byte alto) e riscrivere con PEEK i nuovi valori nelle 3 coppie di puntatori (45/46, 47/48, 49/50). Dopo di ciò si può dare RUN al programma, interromperlo, andare a modificare le istruzioni (basta non allungare troppo per non invadere la zona variabili) senza perdere le variabili. Esse si perdono se dai ancora RUN, ma puoi ripartire con GOTO e portare avanti rapidamente le tue prove. Ricordati però di rimettere a posto le cose dopo.

Un'altro utile suggerimento è quello di sfruttare le possibilità offerte dall'EDITOR per fare poca fatica a scrivere i programmi. Per esempio tutte le linee simili si possono scrivere una dopo l'altra, muovendosi con il cursore e andando a modificare il numero di linea e le parti variate. Attenzione però con le stringhe tra apici, sono un pò delicate da trattare.

Spesso risulta utile fondere dei programmi già esistenti, ma non sempre disponiamo del TOOL KIT che ci servirebbe. Vediamo come si fa ad ottenere la fusione dei programmi (MERGE), con qualche limitazione però. Posssiamo suggerirti due metodi.
Il primo consente di aggiungere a un programma uno o più programmi che però abbiano numeri di linea maggiori in sequenza. Si procede così:
.. carichiamo in memoria il programma con i numeri di linea più bassi;
.. eseguiamo in immediato: PRINT PEEK(45),PEEK(46), ottenendo i byte LO e HI del puntatore all'inizio delle variabili;
.. il numero ottenuto diminuito di 2 unità è l'indirizzo dell'ultimo LINK del programma presente in memoria (i 2 byte a zero che segnalano la fine del programa);
.. scriviamo con POKE i valori dei due byte LO e HI ottenuti nei byte 43 e 44, ottenendo di spostare l'inizio del programma Basic all'indirizzo del LINK nullo precedente;
.. carichiamo con LOAD il programma successivo, quello con i numeri di linea maggiori;
.. ripetiamo la procedura di lettura dei byte 45 e 46 e di riscrittura dei byte 43 e 44 prima descritta fino ad aver caricato tutta la catena dei programmi;
.. alla fine riscriviamo nei byte 43 e 44 i valori iniziali;

.. eseguiamo LIST e vediamo tutto il nostro programma, che può essere memorizzato o su disco o su nastro.

Il secondo metodo consente di fondere programmi anche con numeri di linea che si intrecciano, ma è un pò macchinoso. Supponiamo di voler fondere il programma PRG1 con il programma PRG2. Prendiamo il programmma più corto (PRG2) e lo memorizziamo a pezzi abbastanza corti da poter stare listati su due terzi del video, assegnando ad ogni pezzo un nome. Poi carichiamo il primo pezzo e lo listiamo sul video, carichiamo PRG1, ci portiamo con il cursore sul video e facciamo entrare le linee listate in PRG1 premendo RETURN su ogni linea. A questo punto PRG1 contiene anche un pezzo di PRG2; dobbiamo memorizzarlo, poi richiamare il secondo pezzo di PRG2 e ricominciare. Se sei disposto alla pazienza puoi anche sperimentare questo metodo.

Vediamo ora come si possono cancellare blocchi di linee di programma usando una semplice routine RDEL (cosa necessaria per relizzare il MERGE con il secondo metodo, sopra esposto).

```
50000 REM ROUTINE DELETE
50001 INPUT"DA, A, PASSO";DD,AD,PD
50002 PRINTCHR$(147)
50003 PRINTCHR$(19)DD
50004 DD=DD+PD
50005 PRINT"50010 DD="DD":AD="AD":PD="PD
50006 PRINT"GOTO50010"
50007 POKE631,19:POKE632,13:POKE633,13
50008 POKE634,13:POKE198,4:END
50010 DD= 81::AD= 80::PD= 1
50011 IFDD>ADTHENPRINTCHR$(147):END
50012 GOTO50003
```

Questa routine va aggiunta al programma da modificare con cancellazioni di blocchi di linee. Essa puo essere rilocata cambiando i numeri di linea. Devi farla partire scrivendo GOTO50000. Vediamo cosa succede:

.. alla 50001 viene chiesto da quale linea a quale linea e con quale incremento cancellare;

.. alla 50002 viene eseguito CLR/HOME;

.. alla 50003 viene eseguito HOME e scritto il numero di linea iniziale da cancellare, DD;

.. alla 50004 si incrementa DD con il passo PD;

.. alla 50005 si scrive sul video la linea di programma 50010 che pone le variabili DD, AD e PD ai loro valori assegnati;

.. alla 50006 si scrive sotto sul video GOTO50010;
..alla 50007 si pongono dei caratteri nel buffer della tastiera: HOME, RETURN, RETURN;
.. alla 50008 si pone ancora nel buffer della tastiera un carattere RETURN e si pone il contatore dei caratteri del buffer della tastiera al valore 4 (4 caratteri);
.. il buffer della tastiera si svuota automaticamente eseguendo con il primo RETURN la cancellazione del numero di linea scritto in cima al video, infatti il cursore viene mandato in alto dal carattere HOME che precede il RETURN; gli altri due RETURN fanno accettare la linea 50010 modificata e la fanno eseguire per effetto del GOTO50010;
.. alla 50011 viene controllato che DD non abbia raggiunto il limite AD, nel caso il programma termina;
.. alla 50012 si ha GOTO50003 e si ricomincia.

In questa routine si forza da programma l'intervento dell'EDITOR servendosi del buffer della tastiera.

CAPITOLO 5

STAMPA SU CARTA

5.1 INTRODUZIONE

In questo capitolo facciamo riferimento alla stampante MPS-801 COMMODORE; il calcolatore può essere collegato a una vasta gamma di stampanti. In caso tu ne usi un'altra, dovrai leggere con attenzione il relativo manuale per scoprire se ci sono delle differenze. La maggior parte dei discorsi che facciamo qui restano validi anche per altre stampanti.

La stampante MPS-801 è di tipo grafico; essa può stampare tutti i caratteri del COMMODORE 64, sia quelli del set maiuscolo/grafico, che quelli del set maiuscolo/minuscolo. Essa inoltre consente di definire caratteri particolari, costruendoli per punti, e quindi di produrre grafici.
La stampante dispone di un microprocessore interno che le dà una grande flessibilità. Una ROM interna contiene la mappa dei caratteri, che risulta diversa da quella del calcolatore; i caratteri che appaiono sul video sono costruiti con una matrice di punti 8x8, quelli della stampante con una matrice di punti 6x7. Possiamo scegliere la posizione di inizio stampa sia a livello di carattere (80 per riga) che a livello di punto (480 per riga). Possiamo stampare i caratteri in modo normale, in campo inverso, con ampiezza doppia, e possiamo ripetere colonne di punti. La stampa viene gestita in modo automatico dal microprocessore interno che si serve di un buffer di 90 caratteri situato nella memoria RAM della stampante.
La stampante si collega al calcolatore tramite la porta seriale e devi fare i collegamenti sempre ad apparecchiature spente. Sul retro della stampante trovi un selettore che ti consente di porla nello stato di Self-Test, cioè prova a calcolatore staccato, oppure di selezionare il numero della periferica a 4 o a 5. Nelle prove che seguono supponiamo che il selettore si trovi su 4, cioè usiamo 4 come numero logico della periferica.
Per quanto riguarda i collegamenti, l'inserimento della carta, del nastro e l'aggiustamento della battuta rimandiamo al manuale inserito nella scatola; in esso sono riportate numerose figure esplicative.

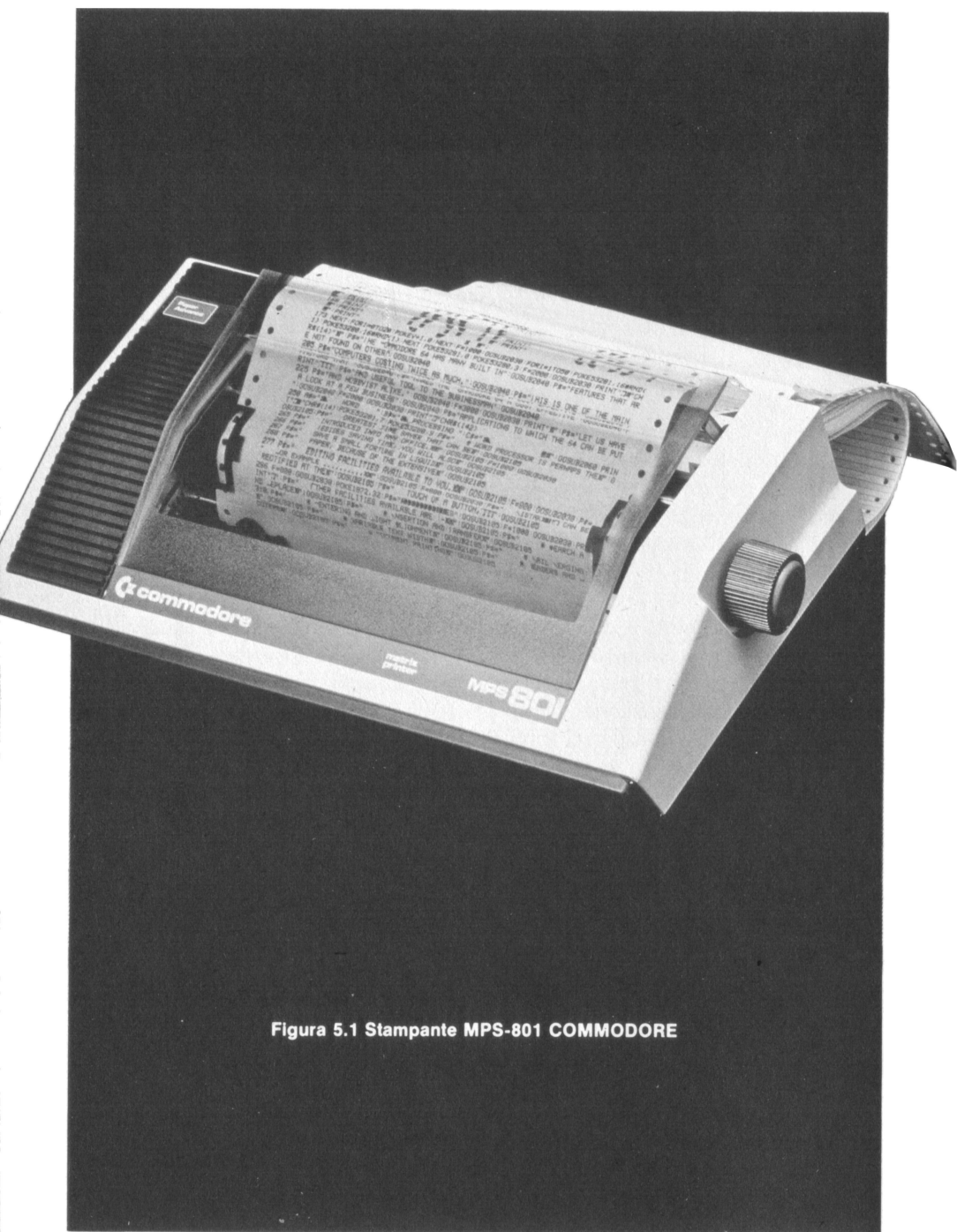



Figura 5.1 Stampante MPS-801 COMMODORE

Riportiamo qui le caratteristiche tecniche di questa stampante:

.. stampa a matrice di punti 6x7
.. caratteri tutti quelli del CBM 64
.. grafici: indirizzabili 7 punti verticali per colonna, con un massimo di 480 colonne
.. codici caratteri ASCII CBM
.. dimensioni carattere: altezza 7 punti (2.82 mm)
larghezza 6 punti (2.53 mm)
.. velocità: 50 car/sec unidirezionale
5 linefeed/sec in modo carattere
7.5 linefeed/sec in modo grafico
.. massimo numero caratteri per linea 80
.. densità stampa: 10 car/pollice
6 linee/pollice in modo carattere
9 linee/pollice in modo grafico
.. larghezza carta: da 4.5 a 10 pollici (comprese le strisce con i fori di trascinamento)
.. numero copie 2
.. nastro a un solo colore con inchiostratore incorporato.

Sul dischetto, contenente i programmi del libro, abbiamo dovuto raggruppare alcuni programmi per non superare la capacità della Directory. Sono stati raggruppati i seguenti programmi:

OPENSA, comprende: OPENSA0, OPENSA7 e LISTASE

e presenta un Menù che permette di scegliere il programma desiderato.

5.2 COMANDI DI STAMPA

Per stampare si deve aprire una via di comunicazione tra il calcolatore e la periferica; questo si ottiene con il comando OPEN, che associa un numero logico di file (lfn) con il numero logico di riconoscimento della periferica (dn).
La sintassi del comando è:

OPEN lfn,dn,sa

dove:

.. lfn (logical file number) è il numero logico del file che si apre; esso può essere un numero compreso tra 1 e 255 (per alcune stampanti i numeri superiori a 127 hanno un particolare significato riguardo al carattere di fine linea); questo numero serve come riferimento in tutti i comandi di stampa;

.. dn (device number) è il numero che individua la periferica e può essere 4 o 5 a seconda della posizione scelta nel selettore posteriore;
.. sa (secondary address) è un numero che stabilisce il modo di operazione della stampante e può assumere i seguenti valori:

.. 0: stampa i caratteri che corrispondono ai codici ricevuti nel set maiuscolo-/grafico
.. 7: stampa i caratteri che corrispondono ai codici ricevuti nel set maiuscolo-/minuscolo.

Il comando di stampa è:

PRINT#lfn,lista-dati

dove:

.. lfn è lo stesso numero usato nella OPEN;
.. lista-dati è l'insieme dei dati da stampare e dei caratteri di controllo per la stampante.

Per scrivere questo comando non si può usare l'abbreviazione del punto interrogativo (?) al posto della parola PRINT e, inoltre NON DEVE esserci spazio tra PRINT e #.
La stampante dopo l'esecuzione del comando PRINT chiude la comunicazione con il calcolatore, cioè non resta in stato di attesa; il file logico resta aperto e viene chiuso solo dalla istruzione CLOSE.
Se lista-dati termina con virgola o punto e virgola la stampante non va a capo. Il punto e virgola non produce spazzi addizionali, mentre la virgola sposta alla prossima zona di stampa (ogni zona di stampa è di 10 caratteri).

Per stampare si può usare anche il comando CMD, che trasferisce l'uscita video alla periferica indicata; in tale caso successivi comandi di PRINT, senza #, continuano a inviare dati alla stampante, che resta in stato di attesa (linea aperta), fino a quando arriva un comando PRINT# per chiudere la linea, e un successivo CLOSE per chiudere il file logico. La sintassi del comando è:

CMD lfn, lista-dati

dove per i parametri vale quanto detto sopra.

Per chiudere il file logico aperto si usa il comando CLOSE la cui sintassi è:

CLOSE lfn

dove: lfn è il numero logico del file.

E' importante chiudere i file aperti quando non servono più, infatti si rischia di riempire la tabella dei file (ne contiene dieci al massimo) e di ricevere una segnalazione di errore.

Riportiamo alcuni semplici esempi circa l'uso dei comandi di stampa, operando sia in modo immediato, che in modo differito.

Seguono tre esempi nei quali usando le istruzioni in modo immediato si ottengono gli stessi risultati, con corretta chiusura della linea e del file.

```
OPEN4,4
PRINT#4,"MPS-801"
CLOSE4
```

```
MPS-801
```

```
OPEN4,4
CMD4,"MPS-801"
PRINT#4:CLOSE4
```

```
MPS-801
```

```
OPEN4,4
CMD4
PRINT"MPS-801"
PRINT#4:CLOSE4
```

```
MPS-801
```

Segueno due esempi di programmi usati in modo differito, nei quali si apre il file di

stampa prima con sa=0 e poi con sa=7, ottenendo la scritta nei due set di caratteri.

```
OPEN CON SA=0



100 REM OPENSA0
105 OPEN1,4,0
110 PRINT#1,"OPEN CON SA=0"
115 CLOSE1

open con sa=0

200 REM OPENSA7
205 OPEN1,4,7
210 PRINT#1,"OPEN CON SA=0"
215 CLOSE1:STOP
```

Segue un esempio di programma che stampa due frasi e poi lista se stesso. Dopo l'esecuzione del comando LIST il controllo non torna più al programma e quindi è necessario scrivere in modo immediato:

PRINT#7:CLOSE7

per chiudere linea e file.

```
PROVA LISTA PROGRAMMA
STAMPA CON CMD

300 REM LISTASE
305 OPEN7,4:CMD7
310 PRINT"PROVA LISTA PROGRAMMA"
315 PRINT"STAMPA CON CMD"
320 LIST300-320
```

Segue il programma SETMPS-801 che stampa le due tabelline dei caratteri dei due set disponibili sulla stampante. La ROM interna alla stampante non è indirizzabile da programma. Il calcolatore invia alla stampante i codici ASCII dei caratteri da

stampare; il microprocessore interno trasforma i codici numerici in caratteri di stampa a matrice di punti 6x7. Quando si stampa in modo grafico, invece, il calcolatore invia alla stampante il codice numerico della colonna di punti da stampare.
Le coordinate orizzontali e verticali nelle due tabelle sono espresse in esadecimale. Le cifre esadecimali da A a F corrispondono ordinatamente ai numeri decimali da 10 a 15. Per ottenere il codice del carattere devi moltiplicare per 16 la coordinata orizzontale (sopra) e aggiungere la coordinata verticale (lato sinistro).

```
1 REM SETMPS-801
5 A$="       SET MAIUSCOLO/GRAFICO"
6 SA=0
10 OPEN4,4,SA:PRINT#4,A$
11 PRINT#4:PRINT#4," | ";
12 FORK=0TO9:PRINT#4,MID$(STR$(K),2);" ";
13 NEXTK
14 FORK=65TO70:PRINT#4,CHR$(K);" ";:NEXTK
15 PRINT#4:PRINT#4,CHR$(192)"┤";
16 FORK=1TO16:PRINT#4,CHR$(192)CHR$(192);
17 NEXTK:PRINT#4:I=0
19 FORK=0TO9:PRINT#4,MID$(STR$(K),2);"| ";
20 GOSUB100:I=I+1:NEXTK
23 FORK=65TO70:PRINT#4,CHR$(K);"| ";
25 GOSUB100:I=I+1:NEXTK
27 IFSA=7THENCLOSE4:STOP
29 PRINT#4:PRINT#4:SA=7
30 A$="       SET MAIUSCOLO/MINUSCOLO"
31 CLOSE4:GOTO10
100 IK=I:FORL=1TO16
101 IFIK<32THENPRINT#4,"  ";:GOTO104
102 IFIK>127ANDIK<160THENPRINT#4,"  ";:GOTO104
103 PRINT#4,CHR$(IK);" ";
104 IK=IK+16:NEXTL:PRINT#4:RETURN
```

RISULTATI PROGRAMMA

SET MAIUSCOLO/GRAFICO

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0				0	@	P	─	┐				┌	─	┐		┌
1			!	1	A	Q	♠	●			▌	┴	♠	●	▌	┴
2			"	2	B	R	│	─			▄	┬	│	─	▄	┬
3			#	3	C	S	─	♥			▔	┤	─	♥	▔	┤
4			$	4	D	T	─	│			▁	│	─	│	▁	│
5			%	5	E	U	─	╮			│	▌	─	╮	│	▐
6			&	6	F	V	─	╳			▒	▐	─	╳	▒	▐
7			'	7	G	W	│	○			│	▔	│	○	│	▔
8			(	8	H	X	│	♣			▒	▀	│	♣	▒	▀
9			)	9	I	Y	╮	│			◤	▄	╮	│	◤	▄
A			*	:	J	Z	╰	♦			│	┘	╰	♦	│	┘
B			+	;	K	[	╯	┼			├	▗	╯	┼	├	▗
C			,	<	L	£	└	▒			▖	▝	└	▒	▖	▝
D			−	=	M	]	╲	│			└	┘	╲	│	└	┘
E			.	>	N	↑	╱	π			┐	▘	╱	π	┐	▘
F			/	?	O	←	┌	◥			▂	▚	┌	◥	▂	π

set maiuscolo/minuscolo

	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
0				0	@	p	─	P				┌	─	P		┌
1			!	1	a	q	A	Q			▌	┴	A	Q	▌	┴
2			"	2	b	r	B	R			▄	┬	B	R	▄	┬
3			#	3	c	s	C	S			▔	┤	C	S	▔	┤
4			$	4	d	t	D	T			▁	│	D	T	▁	│
5			%	5	e	u	E	U			│	▌	E	U	│	▐
6			&	6	f	v	F	V			▒	▐	F	V	▒	▐
7			'	7	g	w	G	W			│	▔	G	W	│	▔
8			(	8	h	x	H	X			▒	▀	H	X	▒	▀
9			)	9	i	y	I	Y			▒	▄	I	Y	▒	▄
a			*	:	j	z	J	Z			│	✓	J	Z	│	✓
b			+	;	k	[	K	+			├	▗	K	+	├	▗
c			,	<	l	£	L	▒			▖	▝	L	▒	▖	▝
d			−	=	m	]	M	│			└	┘	M	│	└	┘
e			.	>	n	↑	N	▚			┐	▘	N	▚	┐	▘
f			/	?	o	←	O	▒			▂	▚	O	▒	▂	▚

Il programma non è molto semplice, ne forniamo una traccia seguendo le linee del listato:

.. in 5 pone A$ uguale alla intestazione della prima tabellina;
.. in 6 pone SA=0, indirizzo secondario per la OPEN per stampare con il set maiuscolo/grafico;
.. in 10 esegue la OPEN della stampante, usando il numero logico di file 4;
.. in 11 stampa l'intestazione;
.. in 12 stampa la prima barrettina verticale e termina con punto e virgola per non andare a capo;
.. in 13 stampa le cifre da 0 a 9, si usa la funzione STR$ per trasformare le cifre in stringa e si prende un solo carattere, facendolo seguire da uno spazio (se si stampa il numero k, esso viene preceduto e seguito da uno spazio e quindi si ottengono due spazi tra le cifre);
.. in 14 si stampano le lettere da A a F separate da uno spazio, usando la funzione CHR$ con argomento il codice della lettera;
.. in 15 e 16 si stampa la sottolineatura con la barretta verticale all'inizio;
.. in 17 si va a capo e si pone I=0;
.. in 19 e 20 si stampano le 10 linee che iniziano con le cifre da 0 a 9, seguite da una barretta verticale e dai caratteri corrispondenti ai codici (questi sono ottenuti con il sottoprogramma che inizia in 100);
.. in 23 e 25 si continua la stampa per le linee che iniziano con le lettere da A a F;
.. in 27 se SA=7 si chiude perchè sono già state stampate le due tabelline;
.. in 29, 30 e 31 si pone SA=7, si prepara la nuova intestazione e si torna in 10 per stampare la seconda tabella.

Le due strisce vuote da 0 a 31 e da 128 a 159 corrispondono ai codici non stampabili.

5.3 MODI DI STAMPA

Alcuni codici, tra quelli non stampabili, quando vengono ricevuti dalla stampante producono modi di stampa diversi. Essi devono essere inviati come caratteri, cioè come stringhe; è comodo, ma non necessario, usare la funzione CHR$.
I codici di controllo in ordine di codice crescente sono:

.. 8 MODO GRAFICO
.. 10 Invio LINE FEED dopo la stampa
.. 13 Invio RETURN dopo la stampa
.. 14 MODO CARATTERE ALLARGATO
.. 15 MODO CARATTERE STANDARD
.. 16 SPOSTAMENTO POSIZIONE STAMPA
.. 17 SET MAIUSCOLO/MINUSCOLO

.. 18 MODO CARATTERE IN CAMPO INVERSO
.. 26 MODO RIPETIZIONE CARATTERE GRAFICO
.. 27 DEFINIZIONE POSIZIONE PUNTO GRAFICO
(deve precedere il codice 16)
.. 145 SET MAIUSCOLO/GRAFICO
.. 146 DISABILITA MODO CARATTERE INVERSO

Vediamo alcuni esempi per chiarire il significato dei codici di controllo della stampa.

CODICE 15

Corrisponde alla stampa normale; deve essere usato per abolire modi prodotti dagli altri codici.

CODICE 14

Usando questo codice si ottiene di allargare i caratteri. Il modo di stampa normale (standard) corrisponde all'uso del codice di controllo 15. Devi fare attenzione; dopo aver usato un codice di controllo che disabilita il modo normale, devi usare il codice 15 per rimettere tutto a posto.

Segue il programma CHR14-1, che stampa due linee allargate, poi torna al modo normale e lista se stesso. Alla fine devi eseguire in immediato:

PRINT#10:CLOSE10

per chiudere correttamente.

```
COMMODORE
 MPS-801


1 REM CHR$(14)-1
10 OPEN10,4
15 CMD10
20 PRINTCHR$(14)"COMMODORE"
25 PRINTCHR$(14)" MPS-801 "
30 PRINTCHR$(15)
40 LIST
```

Segue il programma CHR14-2 che stampa sulla stessa riga caratteri allargati e caratteri normali.

```
SEARCHING FOR CHR14-2CHR14-2

1 REM CHR$(14)-2
10 OPEN10,4
20 PRINT#10,CHR$(14)"COMMODORE";
23 PRINT#10,CHR$(15)"COMMODORE"
25 PRINT#10,CHR$(14)" MPS-801 ";
27 PRINT#10,CHR$(15)" MPS-801 "
29 FORK=65TO68
31 PRINT#10,CHR$(14)CHR$(K)CHR$(15)CHR$(K);
33 NEXTK
50 PRINT#10,CHR$(15):CLOSE10
```

RISULTATI PROGRAMMA CHR14-2

```
COMMODORECOMMODORE
 MPS-801   MPS-801
AABBCCDD
```

CODICE 8

E' il codice per stampare in modo grafico, cioè fornendo alla stampante il codice numerico della colonna di punti da stampare; essa è formata da 7 punti.

Il codice numerico si ottiene attribuendo ai 7 punti incolonnati i seguenti pesi, dall'alto verso il basso: 1, 2, 4, 8, 16, 32, 64.
Per disegnare caratteri grafici devi usare un foglio di carta a quadretti e dividerlo in strisce alte 7 quadretti. In ogni striscia puoi disegnare i caratteri che desideri; poi applicando i pesi calcolare il numero che corrisponde a ogni colonna.

Vediamo come esempio la costruzione del carattere che abbiamo usato nei programmi che seguono.

Carattere desiderato:

```
*       *   *       *
 *     * *         *
  *     *         *
   *   * *       *
    *   *   *   *
     * *     * *
      *       *
```

esso occupa 4 caratteri di stampa 6x7 per nostra decisione, potevamo infatti usare 3 colonne di punti in meno; in tale caso stampando più caratteri vicini verrebbero attaccati.
Ora disegnamo il carattere usando 0 e 1 e riportando sulla sinistra i pesi da attribuire ad ogni riga:

```
1       010000000100010000000010
2       001000000010100000000100
4       000100000000100000001000
8       000010000010100000010000
16      000001000100010001000000
32      000000101000001010000000
64      000000010000000100000000
```

applicando i pesi e facendo le somme per colonna otteniamo:

```
0x1+0x2+0x4+0x8+0x16+0x32+0x64=   0
1x1+0x2+0x4+0x8+0x15+0x32+0x64=   1
0x1+1x2+0x4+0x8+0x16+0x32+0x64=   2
0x1+0x2+1x4+0x8+0x16+0x32+0x64=   4
0x1+0x2+0x4+1x8+0x16+0x32+0x64=   8
0x1+0x2+0x4+0x8+1x16+0x32+0x64=  16
0x1+0x2+0x4+0x8+0x16+1x32+0x64=  32
0x1+0x2+0x4+0x8+0x16+0x32+1x64=  64
0x1+0x2+0x4+0x8+0x16+1x32+0x64=  32
1x1+0x2+0x4+0x8+1x16+0x32+0x64=  17
0x1+1x2+0x4+1x8+0x16+0x32+0x64=  10
0x1+0x2+1x4+0x8+0x16+0x32+0x64=   4
0x1+1x2+0x4+1x8+0x16+0x32+0x64=  10
1x1+0x2+0x4+0x8+1x16+0x32+0x64=  17
0x1+0x2+0x4+0x8+0x16+1x32+0x64=  32
0x1+0x2+0x4+0x8+0x16+0x32+1x64=  64
0x1+0x2+0x4+0x8+0x16+1x32+0x64=  32
0x1+0x2+0x4+0x8+1x16+0x32+0x64=  16
0x1+0x2+0x4+1x8+0x16+0x32+0x64=   8
0x1+0x2+1x4+0x8+0x16+0x32+0x64=   4
0x1+1x2+0x4+0x8+0x16+0x32+0x64=   2
1x1+0x2+0x4+0x8+0x16+0x32+0x64=   1
0x1+0x2+0x4+0x8+0x16+0x32+0x64=   0
0x1+0x2+0x4+0x8+0x16+0x32+0x64=   0
```

I numeri scritti a destra sono i codici da passare alla stampante dopo averli sommati a 128.

Segue il programma CHR8-1, nel quale si fanno entrare in 4 linee DATA i codici numerici che definiscono il carattere visto sopra, poi si prelevano uno dopo l'altro, e, dopo avergli sommato 128 si trasformano in stringa. Si raggruppano 6 caratteri in ogni stringa formando 4 stringhe A$(i). Non è necessario raggruppare i codici a 6 a 6; essi hanno validità colonna per colonna. Il programma stampa in sequenza 5 caratteri sulla stessa riga, poi 7, poi 9 e poi 11, iniziando sempre dal margine sinistro.

```
1 REM CHR$(8)-1
10 OPEN10,4,7
15 CMD10
20 DATA0,1,2,4,8,16
21 DATA32,64,32,17,10,4
22 DATA10,17,32,64,32,16
23 DATA8,4,2,1,0,0
30 FORI=1TO4:A$(I)="":FORJ=1TO6
40 READA:A=A+128
45 A$(I)=A$(I)+CHR$(A)
50 NEXTJ:NEXTI
51 FORK=1TO5
53 PRINTCHR$(8)A$(1)A$(2)A$(3)A$(4);
54 NEXTK:PRINT
55 FORK=1TO7
57 PRINTCHR$(8)A$(1)A$(2)A$(3)A$(4);
59 NEXTK:PRINT
61 FORK=1TO9
63 PRINTCHR$(8)A$(1)A$(2)A$(3)A$(4);
65 NEXTK:PRINT
67 FORK=1TO11
69 PRINTCHR$(8)A$(1)A$(2)A$(3)A$(4);
71 NEXTK:PRINT
75 PRINT:PRINTCHR$(15)
80 PRINT#10:CLOSE10
```

RISULTATI PROGRAMMA

CODICE 26

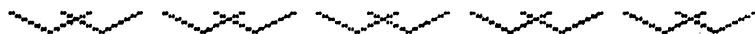
Questo codice permette di ripetere un determinato carattere grafico, inteso come

colonna di punti, per un definito numero di volte. Per ottenere questo risultato si deve:

.. entrare in modo grafico con il codice 8;
.. entrare in modo ripetizione con il codice 26;
.. inserire il numero N di ripetizioni, da 0 a 255 (1 byte), fornendolo come stringa (CHR$(N));
.. fornire la stringa che dà il codice della colonna di punti.

Segue il programma CHR26-1, nel quale il carattere speciale da noi definito viene stampato 5 volte ripetendo due volte ogni colonna di punti; in conseguenza il carattere risulta allargato. Nota la linea 33 con la sequenza: CHR$(8)CHR$(26)CHR$(2)A$.

```
1 REM CHR$(26)-1
10 OPEN10,4
15 CMD10
20 DATA0,1,2,4,8,16
21 DATA32,64,32,17,10,4
22 DATA10,17,32,64,32,16
23 DATA8,4,2,1,0,0
25 FORK=1TO5
30 FORI=1TO24
31 READA:A$=CHR$(A+128)
33 PRINTCHR$(8)CHR$(26)CHR$(2)A$;
35 NEXTI
37 RESTORE:NEXTK
75 PRINT:PRINTCHR$(15)
80 PRINT#10:CLOSE10
```

RISULTATI PROGRAMMA

La possibilità di ripetere colonne di punti è molto utile per tracciare istogrammi. Segue il programma CHR26-2, nel quale si definisce un carattere G$ formato da una colonna di punti con "accesi" solo 5 punti (252, manca il punto corrispondente al peso 2 e quello corrispondente al peso 1), in tale modo le parti del grafico non si toccano. Inoltre con un DATA sono memorizzati 10 numeri relativi all'andamento delle vendite nel decennio 1974/1983. Il programma stampa il grafico che visualizza l'andamento delle vendite.
La limitazione sul numero N di ripetizioni minori di 256 può essere superata

ripetendo due volte di seguito l'operazione, senza dimenticare che il numero massimo di punti su una riga è 480. In caso si possono ridurre i fattori di scala per rientrare nelle dimensioni consentite.

```
1 REM CHR$(26)-2
5 OPEN10,4,7
10 CMD10:G$=CHR$(252)
15 DATA18,20,22,30,50,48,47,49,60,28
20 PRINTCHR$(14)"ANDAMENTO VENDITE"
23 PRINT"DECENNIO 1974/1983"
25 PRINT
30 FORI=1TO10
35 READB
40 C=1973+I
45 PRINTCHR$(15)C;"    ";CHR$(8)CHR$(26)CHR$(B)G$
50 NEXTI
55 PRINT#10,CHR$(15):CLOSE10:STOP
```

```
andamento vendite
deciannio 1974/1983

1974
1975
1976
1977
1978
1979
1980
1981
1982
1983
```

```
ANDAMENTO VENDITE
DECENNIO 1974/1983

1974
1975
1976
1977
1978
1979
1980
1981
1982
1983
```

Abbiamo riportato due esempi di risultato per il programma precedente; il primo è stato ottenuto usando il programma come nel listato con SA=7 nella OPEN alla linea 5, il secondo ponendo SA=0 o senza SA. La differenza si ha solo nella intestazione che appare con i caratteri del set selezionato, infatti il set attivo non influisce sui caratteri stampati in modo grafico per punti e sulla stampa dei numeri.

CODICE 16

Questo codice serve per fissare la colonna di inizio stampa tra 0 e 79; il numero della colonna deve essere dato dalle 2 cifre passate come 2 caratteri stringa, quindi o come CHR$(N) con N che varia da 48 a 57 oppure come "nn".

Nel programma CHR16-1 si stampa una riga di riferimento numerata per segnare 40 colonne di stampa, poi sotto si stampa 3 volte una cifra seguita dal carattere speciale usato nei nostri esempi. I codici delle cifre che danno la colonna sono passati con un DATA alla linea 27; le colonne sono 00 (48, 48), 15 (49, 53) e 30 (51, 48). Vedi che 0 è stampato alla colonna 1, infatti il formato di stampa del numero è: uno spazio, le cifre, uno spazio. Osserva la linea 55; dopo le, diciamo, coordinate della colonna X e Y si pone la variabile I che contiene il numero da stampare, poi si passa al modo grafico con CHR$(8) e si stampa il carattere speciale senza spostare ulteriormente la posizione di stampa.

```
1 REM CHR$(16)-1
10 OPEN10,4
15 CMD10:A$=""
20 DATA0,1,2,4,8,16,32,64,32,17,10,4
25 DATA10,17,32,64,32,16,8,4,2,1,0,0
27 DATA48,48,49,53,51,48
30 FORI=1TO24
35 READA
40 A$=A$+CHR$(A+128)
45 NEXTI
47 FORK=0TO4:FORI=0TO9:PRINTCHR$(48+I);
48 NEXTI:NEXTK:PRINT
50 FORI=0TO2:READX,Y
55 PRINTCHR$(15)CHR$(16)CHR$(X)CHR$(Y);I;CHR$(8)A$;
60 NEXTI
65 PRINT#10,CHR$(15):CLOSE10:STOP
```

RISULTATI PROGRAMMA

```
0123456789012345678901234567890123456789
 0 \/X\/              1 \/X\/              2 \/X\/
```

CODICE 27

Questo codice consente, se usato prima del codice 16, di spostare l'inizio della posizione di stampa in uno dei possibili 480 punti di una riga (si conta da 0 a 479). Ovviamente deve essere usato dopo essere entrati in modo grafico, cioè stampa a punti e non a caratteri. Per usare correttamente il codice si deve procedere così:

.. entrare in modo grafico con il codice 8;
.. entrare in modo indirizzo per punti con il codice 27;
.. entrare in modo TAB con il codice 16;
.. scrivere due caratteri stringa che siano i caratteri corrispondenti al valore numerico del byte HI e del byte LO della distanza in punti dall'inizio della riga (si considera il numero dei punti N e si pone nel primo byte il numero ottenuto con INT(N/256) e nel secondo N-INT(N/256), poi si opera con la funzione CHR$ su questi valori);
.. scrivere i caratteri (sotto forma di stringa) che danno le colonne di punti da stampare.

Segue il programma CHR27-1 nel quale viene stampato lo stesso insieme di caratteri del programma CHR8-1, solo che ogni riga inizia in una posizione diversa. La prima riga inizia nel punto 72, la seconda nel punto 48, la terza hel punto 24 e la quarta nel punto 0. Ricorda che ci vogliono sempre i due byte HI e LO, anche se il primo corrisponde a zero.

```
1 REM CHR$(27)-1
10 OPEN10,4,7
15 CMD10
20 DATA0,1,2,4,8,16
21 DATA32,64,32,17,10,4
22 DATA10,17,32,64,32,16
23 DATA8,4,2,1,0,0
30 FORI=1TO4:A$(I)="":FORJ=1TO6
40 READA:A=A+128
45 A$(I)=A$(I)+CHR$(A)
50 NEXTJ:NEXTI
51 FORK=0TO4
52 PRINTCHR$(8)CHR$(27)CHR$(16)CHR$(0);
53 PRINTCHR$(24*(3+K))A$(1)A$(2)A$(3)A$(4);
54 NEXTK:PRINT
55 FORK=0TO6
57 PRINTCHR$(8)CHR$(27)CHR$(16)CHR$(0);
58 PRINTCHR$(24*(2+K))A$(1)A$(2)A$(3)A$(4);
```

```
59 NEXTK:PRINT
61 FORK=0TO8
62 PRINTCHR$(8)CHR$(27)CHR$(16)CHR$(0);
63 PRINTCHR$(24*(1+K))A$(1)A$(2)A$(3)A$(4);
65 NEXTK:PRINT
67 FORK=0TO10
69 PRINTCHR$(8)A$(1)A$(2)A$(3)A$(4);
71 NEXTK:PRINT
75 PRINT:PRINTCHR$(15)
80 PRINT#10:CLOSE10
```

RISULTATI PROGRAMMA

In realtà nel programma facciamo una cosa inutile, cioè nella stampa ciclica dei caratteri di ogni linea rifacciamo il posizionamento a livello di carattere speciale; sarebbe stato sufficiente posizionarsi all'inizio per il primo carattere e poi proseguire semplicemente con gli altri.
Nota la differenza di uso del carattere 16; se da solo deve essere seguito dalle due cifre della colonna passate come stringa, se preceduto da 27, deve essere seguito dai caratteri ricavati dai due byte binari.

Nel programma CHR27-2 invece si stampano 5 tacche nelle posizioni punto 0, 50, 100, 150 e 200.

```
1 REM CHR$(27)-2
10 OPEN10,4
15 CMD10
20 FORI=0TO4
30 A=50*I
40 B=INT(A/256)
50 C=A-B*256
60 PRINTCHR$(8)CHR$(27)CHR$(16);
65 PRINTCHR$(B)CHR$(C)CHR$(255);
70 NEXTI:PRINT#10,CHR$(15):CLOSE10
```

RISULTATI PROGRAMMA

| | | | |

CODICE 17

Questo codice fa cambiare il set di caratteri della stampante passando al modo maiuscolo/minuscolo. Esso viene annullato solo dal codice 145 che fa tornare all'altro set, che è quello disponibile al momento dell'accensione.

Seguono tre programmi esempio.

```
commodore



1 rem chr$(17)-1
10 open10,4,7
20 cmd10
30 printchr$(17)"commodore"
40 printchr$(15):print
50 list

commodore

commodore

1 rem chr$(17)-3
10 open10,4
20 cmd10
30 printchr$(17)"commodore"
40 printchr$(15):print"commodore"
50 list

commodore

commodore

1 rem chr$(17)-2
10 open10,4,7
20 cmd10
30 print"commodore"
40 printchr$(15):print"commodore"
50 list
```

Come vedi in CHR17-1 si apre con SA=7 e poi si usa anche il codice 17, mentre se ne potrebbe fare a meno. L'uso del codice 15 non ha effetto sul set di caratteri e il comando LIST fa listare il programma senza cambiare caratteri.
Nel programma CHR17-2 non si usa il codice 17, anche qui il codice 15 non ha effetto; la OPEN ha SA=7.
Nel programma CHR17-3 non si usa SA=7, ma usando il codice 17 si ottengono gli stessi risultati.
Dato che i tre programmi terminano con LIST, dopo devi scrivere in immediato: PRINT#10:CLOSE10.

CODICE 145

Questo codice fa tornare al set di caratteri maiuscolo/grafico.
Segue il programma CHR145-1 a dimostrare quanto detto. Anche questo termina con LIST e quindi devi chiudere in immediato.

```
commodore

COMMODORE

800 REM CHR$(145)-1
802 OPEN10,4
804 CMD10
806 PRINTCHR$(17)"COMMODORE"
808 PRINTCHR$(145):PRINT"COMMODORE"
810 LIST
```

CODICI 18 E 146

Questi codici servono per passare in campo inverso e per tornare in campo diretto.
Segue il programma CHR18-CHR146 che esemplifica l'uso dei due codici.

```
1 REM CHR$(18)-CHR$(146)
10 A$="CARATTERE NORMALE"
15 B$="CARATTERE IN CAMPO INVERSO"
20 C$="CAMBIO"
30 OPEN4,4:CMD4
35 PRINTA$
40 PRINTCHR$(18)B$CHR$(146)
50 FORK=1TO2
55 PRINTCHR$(18)C$CHR$(146)C$;
60 NEXTK:PRINT#4
65 CLOSE4:STOP
```

RISULTATI PROGRAMMA CHR18-CHR146

```
CARATTERE NORMALE
[illegible]
[illegible]CAMBIO[illegible]CAMBIO
```

Dagli esempi di questo capitolo risulta chiaramente il fatto che usare uno o l'altro set di caratteri in stampa o il campo diretto/inverso dipende dai codici di controllo passati alla stampante.
Per chiarire ulteriormente l'argomento facciamo seguire il programma GETPEEK, nel quale alla linea 10 viene scritto sulla prima riga del video la parola PIPPO in campo diretto e inverso; all'interno della stringa di stampa la parola da stampare in campo inverso è preceduta da RVSON e seguita da RVSOFF. Dopo, alla linea 15 vengono prelevati dal video con il comando PEEK i 10 caratteri scritti e memorizzati in A(K). Alla linea 20 si porta il cursore all'inizio del video senza cancellarlo e si apre il video come file, poi alla 25 si leggono con 10 GET#3 i caratteri scritti e si memorizzano in A$(K). Alla 30 si usa il "file video" già aperto per scrivere con PRINT#3 e si scende di 8 linee, poi si stampa A(K) e A$(K). Alla 45 si chiude il video; nelle linee seguenti si apre la stampante e si stampano i risultati.

```
1 REM GETPEEK
10 PRINT"[illegible]PIPPO[illegible]PIPPO[illegible]"
15 FORK=0TO9:A(K)=PEEK(1024+K):NEXTK
20 PRINT"[illegible]";:OPEN3,3
25 FORK=0TO9:GET#3,A$(K):NEXTK
30 PRINT#3,"[illegible]";
35 FORK=0TO9:PRINT#3,A(K);:NEXTK:PRINT#3
40 FORK=0TO9:PRINT#3,ASC(A$(K));:NEXTK
45 PRINT#3:CLOSE3
47 OPEN4,4:PRINT#4,"PIPPO DIRETTO E INVERSO
49 PRINT#4,"VALORI LETTI CON PEEK: "
50 FORK=0TO9:PRINT#4,A(K);:NEXTK:PRINT#4
51 PRINT#4,"VALORI LETTI CON GET: "
55 FORK=0TO9:PRINT#4,ASC(A$(K));:NEXTK
57 PRINT#4:CLOSE4:STOP
```

RISULTATI PROGRAMMA GETPEEK

```
PIPPO DIRETTO E INVERSO
VALORI LETTI CON PEEK:
 16  9  16  16  15  144  137  144  144  143
VALORI LETTI CON GET:
 80  73  80  80  79  80  73  80  80  79
```

Come puoi vedere dai risultati i 10 valori letti con PEEK sono i D/CODE dei caratteri letti, e in essi è riconoscibile il campo inverso (codice +128: 16+128=144). Nei 10 valori successivi, quelli letti con GET#3 si vedono solo i codici ASCII dei caratteri letti e non si ha differenza tra quelli diretti e quelli inversi. I codici passati alla stampante non sono i D/CODE, ma gli ASCII, quindi se si vuole il campo inverso si deve comandare con un codice di controllo.

CODICI 10 E 13

Questi codici producono lo stesso effetto; si ha un effetto diverso a fine linea in dipendenza del numero logico usato per aprire il "file stampante". Segue un programma dimostrativo.

```
1 REM CHR$(10)CHR$(13)
10 OPEN129,4
20 A$="PROVA1 LFN>128":B$="PROVA DI CHR$(10)"
25 PRINT#129,A$CHR$(10)B$
30 C$="PROVA2 LFN>128":D$="PROVA DI CHR$(13)"
35 PRINT#129,C$CHR$(13)D$
40 CLOSE129
50 OPEN10,4
60 E$="PROVA3 LFN<128":F$="PROVA DI CHR$(10)"
65 PRINT#10,E$CHR$(10)F$
70 G$="PROVA4 LFN<128":H$="PROVA DI CHR$(13)"
75 PRINT#10,G$CHR$(13)H$
80 CLOSE10
90 STOP
```

RISULTATI PROGRAMMA

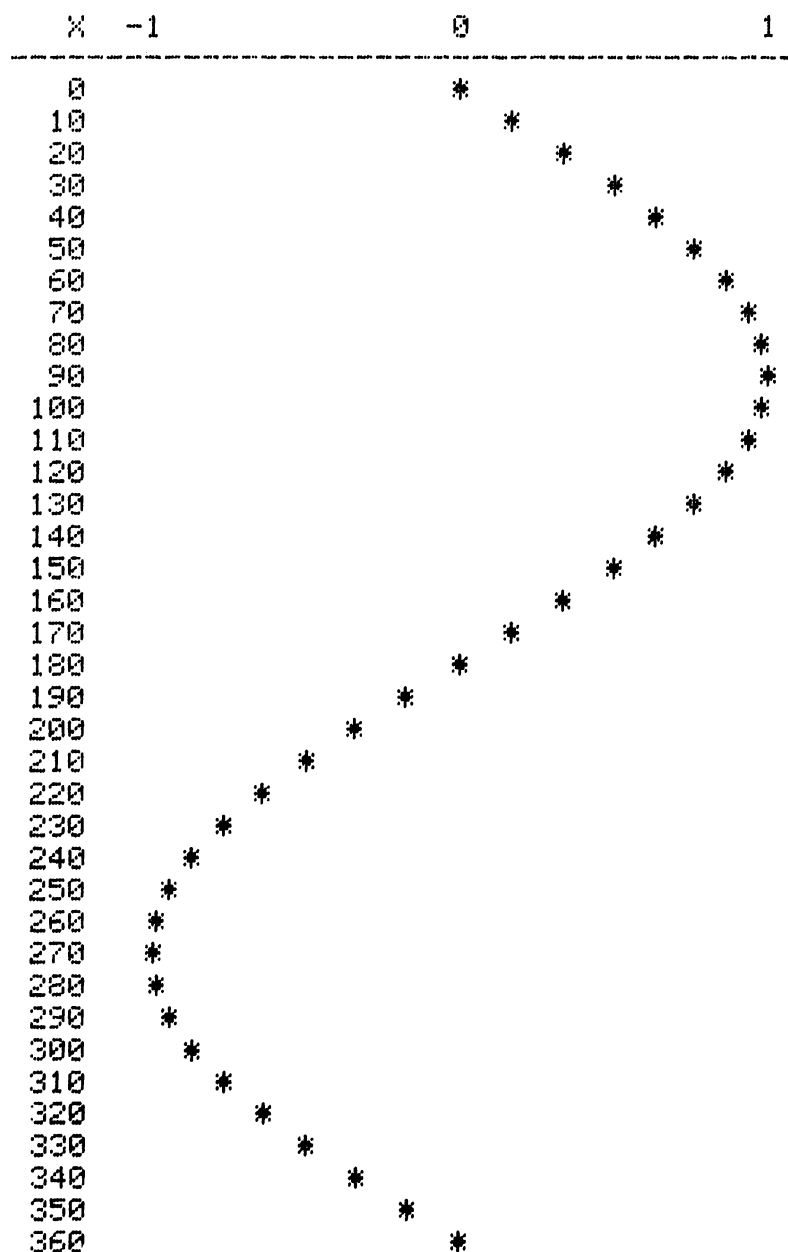
```
PROVA1 LFN>128
PROVA DI CHR$(10)

PROVA2 LFN>128
PROVA DI CHR$(13)

PROVA3 LFN<128
PROVA DI CHR$(10)
PROVA4 LFN<128
PROVA DI CHR$(13)
```

Come vedi, quando si usa LFN=129 (maggiore di 128), a fine linea si va a capo due volte, mentre il 10 e il 13 producono lo stesso effetto, mandano a capo una volta. Quando, invece, si usa LFN < 128, resta invariato il comportamento dei codici, ma a fine linea si va a capo 1 sola volta.

Per chiudere questo paragrafo riportiamo il programma GRAF1 che produce il grafico della funzione SIN tra 0 e 360.

```
1 REM GRAF1
5 OPEN4,4:CMD4
10 D$=CHR$(14):N$=CHR$(15)
15 P$=CHR$(16):G$=CHR$(27)
20 C=23:A=16:O=4
25 A$="-":FORI=0TOC+A:A$=A$+"-":NEXT
30 S$="       "
35 PRINTD$"  GRAFICO SIN"
40 PRINTN$
45 PRINTLEFT$(S$,O-1)+"X";
50 PRINTSPC(C-A-O-1)"-1";
55 PRINTSPC(A-1)"0";
60 PRINTSPC(A-1)"1"
65 PRINTA$
70 FORI=0TO360STEP10
75 I$=RIGHT$(S$+STR$(I),O)
80 YO=C*6+A*6*SIN(I*π/180)
85 YH=INT(YO/256):YL=YO-YH*256
90 PRINTI$G$P$CHR$(YH)CHR$(YL)"*"
95 NEXTI:PRINT#4,N$:CLOSE4:STOP
```

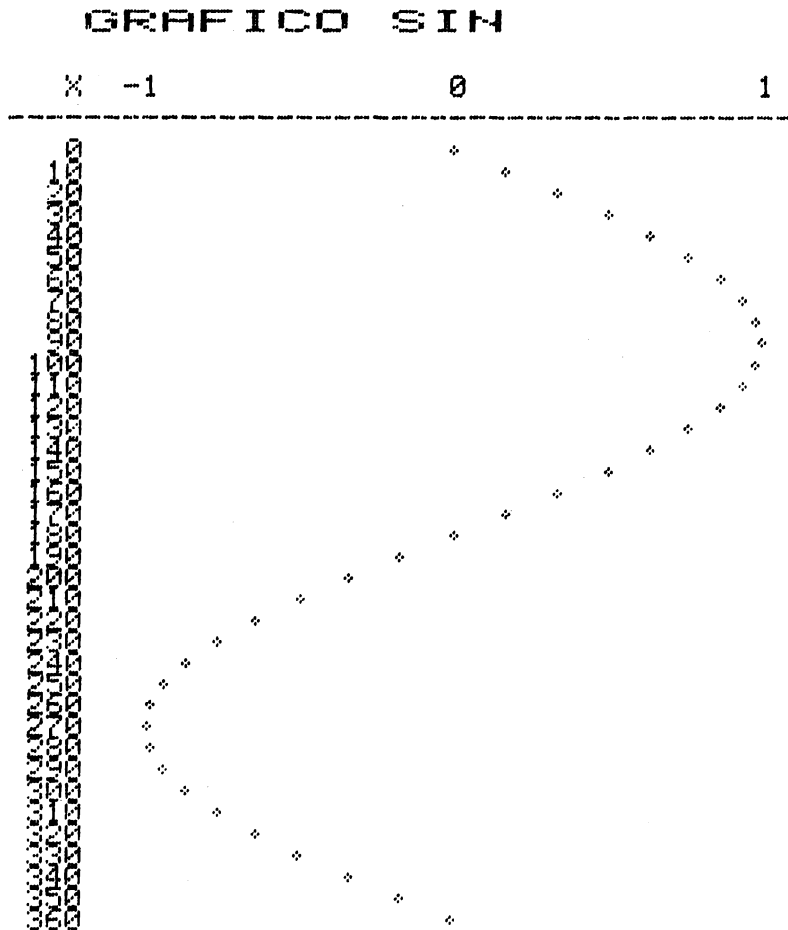
RISULTATI PROGRAMMA GRAF1

```
      GRAFICO  SIN

  X  -1                0                 1
--------------------------------------------
  0                    *
 10                       *
 20                          *
 30                             *
 40                               *
 50                                 *
 60                                   *
 70                                    *
 80                                     *
 90                                      *
100                                     *
110                                    *
120                                   *
130                                 *
140                               *
150                             *
160                          *
170                       *
180                    *
190                 *
200              *
210            *
220         *
230       *
240     *
250    *
260   *
270   *
280   *
290    *
300     *
310       *
320         *
330            *
340              *
350                 *
360                    *
```

Nota che i codici di controllo sono stati inizialmente assegnati a delle variabili stringa. Si usa la funzione SPC per ottenere gli spazi e si fa un uso diverso da quello visto prima della sequenza di codici 27 e 16 (G$ e P$) per ottenere la posizione di stampa pur non essendo in modo grafico (codice 8).

Segue il programma GRAF2 che abbiamo ottenuto modificando leggermente GRAF1, e cioè passando a stampare la funzione in modo grafico con il carattere F$ formato da tre colonne di punti (un piccolo rombo). Nota che il grafico risulta più corto; infatti la stampante va a capo mentre si trova in modo grafico e in questo modo si hanno 9 linee per pollice invece di 6.

```
1 REM GRAF2
5 OPEN4,4:CMD4
10 D$=CHR$(14):N$=CHR$(15):GR$=CHR$(8)
15 P$=CHR$(16):G$=CHR$(27):NO$=CHR$(15)
20 C=23:A=16:O=4:F$=CHR$(136)+CHR$(148)+CHR$(136)
25 A$="-":FORI=0TOC+A:A$=A$+"-":NEXT
30 S$="          "
35 PRINTD$"  GRAFICO SIN"
40 PRINTN$
45 PRINTLEFT$(S$,O-1)+"X";
50 PRINTSPC(C-A-O-1)"-1";
55 PRINTSPC(A-1)"0";
60 PRINTSPC(A-1)"1"
65 PRINTA$
70 FORI=0TO360STEP10
75 I$=RIGHT$(S$+STR$(I),O)
80 YO=C*6+A*6*SIN(I*π/180)
85 YH=INT(YO/256):YL=YO-YH*256
90 PRINTNO$I$GR$G$P$CHR$(YH)CHR$(YL)F$
95 NEXTI:PRINT#4,N$:CLOSE4:STOP
```

RISULTATI PROGRAMMA GRAF2

```
GRAFICO SIN

X  -1              0              1
-----------------------------------------
```

5.4 STAMPA AUTOMATICA

Il buffer della stampante contiene 90 caratteri; quando si manda qualcosa alla stampante, i caratteri sono ammucchiati nel buffer, siano essi caratteri da stampare o caratteri di controllo. La stampa avviene in modo automatico quando il buffer è pieno e può dar luogo allo svuotamento totale o parziale del buffer.
La stampa avviene nelle seguenti condizioni:

.. Il buffer è pieno, cioè contiene 90 caratteri, ma i caratteri da stampare (togliendo cioè quelli di controllo) non raggiungono gli 80 (o i 480 punti in modo grafico). In questo caso il buffer viene svuotato completamente, si ha la stampa di una linea senza andare a capo (infatti non è stato incontrato un carattere di codice 10 o 13 che fa andare a capo).

.. Il buffer contiene meno di 90 caratteri, ma riceve un comando di vai a capo, si ha la stampa di tutti i caratteri con svuotamento completo del buffer e si va a capo.

.. Il buffer non è pieno ma contiene più caratteri o punti di quelli che stanno su una linea. Si ha la stampa di una linea con a capo, il buffer viene svuotato solo dei caratteri stampati.

In sostanza quando stampi a volte vedrai uscire con ritardo quello che ti aspetti e a volte NON vedrai uscire gli ultimi caratteri quando chiudi il file, infatti la CLOSE non svuota il buffer (conviene terminare sempre con una PRINT senza lista di dati così il buffer si vuota di sicuro).

Il comportamento descritto è valido quando si usa la stampante aprendola come file con i comandi di stampa PRINT#. Nel caso del CMD si può avere un comportamento diverso; infatti quando il calcolatore manda un READY o un altro messaggio, che non esce più sul video, la stampante va necessariamente a capo e stampa il messaggio.

Ricorda che se la lista dei dati da stampare non termina con virgola o punto e virgola si ha l'aggiunta automatica di un CHR$(13) e quindi un a capo. La virgola fa saltare a completamento di multipli di 10 spazi, come sul video. Il punto e virgola non fa spaziare.

Segue un programma per confermare quanto detto.

```
1 REM ST-AUTOM
10 OPEN4,4
15 LF$=CHR$(10):CR$=CHR$(13)
20 A$="0123456789":B$=""
25 A1$="ABCDEFGHIJKLMNOPQRSTUVWXYZ":B1$=""
30 FORK=1TO8:B$=B$+A$:NEXTK
35 FORK=1TO4:B1$=B1$+A1$:NEXTK
40 PRINT#4,B$:GOSUB1000
50 PRINT#4,B$;A$:GOSUB1000
60 PRINT#4,B$;A$;:GOSUB1000
70 PRINT#4,LF$;LF$;LF$;LF$;LF$;LF$;LF$;
```

```
71 PRINT#4,LF$;LEFT$(B$,70);CR$;
75 PRINT#4,CR$;CR$;CR$;CR$;CR$;CR$;CR$;
76 PRINT#4,CR$;CR$;CR$;CR$;CR$;A1$:GOSUB1000
80 PRINT#4,LF$;LF$;LF$;LF$;LF$;LF$;LF$;
81 PRINT#4,LF$;LEFT$(B1$,70);CR$;
85 PRINT#4,CR$;CR$;CR$;CR$;CR$;CR$;CR$;
86 PRINT#4,CR$;CR$;CR$;CR$;CR$;A$;:GOSUB1000
999 CLOSE4:STOP
1000 FORI=1TO1000:NEXTI:RETURN
```

In ST-AUTOM vengono preparate le seguenti stringhe:
.. A$ con le 10 cifre decimali;
.. B$ con le 10 cifre ripetute 8 volte;
.. A1$ con le 26 lettere dell'alfabeto;
.. B1$ con le 26 lettere ripetute 4 volte per un totale di 104 caratteri;
.. LF$ con il carattere LINE FEED;
.. CR$ con il carattere RETURN.

Alla linea 1000 si ha un sottoprogramma che crea un breve ciclo di attesa per consentire di osservare i risultati della stampante. Lasciamo a te lo studio dei risultati che ottieni. Alla fine ricordati di eseguire in immediato:

OPEN4,4:PRINT#4:CLOSE4

questo per ottenere la stampa degli ultimi 10 caratteri rimasti nel buffer.

5.5 FORMATO DEI TABULATI

Nella preparazione di stampati si presentano spesso problemi di incolonnamento e allineamento dei dati. Il sistema trasforma tutto quello che stampa in stringhe di caratteri, ma mentre se tu operi con STR$ su un numero ottieni le cifre più uno spazio o il segno meno prima, e niente dopo, quando lo fa implicitamente il sistema esso aggiunge anche uno spazio dopo. Per ottenere buoni tabulati si lavora con la funzione SPC, con i comandi LEFT$, MID$ e RIGHT$, con la funzione LEN. La funzione TAB agisce anche per la stampante, ma in modo diverso che sul video, cioè non conta le posizioni da inizio riga, ma dall'ultima posizione di stampa, cioè agisce come SPC. Inoltre per gli allineamenti orizzontali si può usare il codice di controllo 16. In generale si desidera che i numeri siano allineati a destra e le parole a sinistra.
Altro problema è quello delle spaziature orizzontali; in generale si istituisce un contarighe, che viene azzerato ad ogni inizio di pagina ed aggiornato per ogni riga

stampata. Quando si vuole andare a nuovo foglio basta stampare a vuoto per tante righe quante mancano al raggiungimento del valore limite del contatore (numero linee del foglio). Alcune stampanti hanno il contarighe automatico ed accettano un codice di controllo che fa andare a nuovo foglio, questa no.

Seguono alcuni esempi relativi a quanto detto.

Nei due programmi INC1 e INC2 si usano le funzioni TAB e SPC per ottenere di stampare in determinate posizioni; all'inizio si stampa una linea di numeri di riferimento.

```
1 REM INC1
10 OPEN4,4:CMD4
20 A$="0123456789":B$=""
30 FORK=1TO4:B$=B$+A$:NEXTK
40 PRINTB$
50 PRINTTAB(0)"POS0";TAB(6)"POS10";
55 PRINTTAB(15)"POS30"
60 PRINTTAB(80);"NUOVA RIGA"
100 PRINT#4:CLOSE4:STOP
```

RISULTATI PROGRAMMA INC1

```
0123456789012345678901234567890123456789
POS0      POS10               POS30

NUOVA RIGA
```

RISULTATI PROGRAMMA INC2

```
1 REM INC2
10 OPEN4,4:CMD4
20 A$="0123456789":B$=""
30 FORK=1TO4:B$=B$+A$:NEXTK
40 PRINTB$
50 C$="DIECI CAR."
60 PRINTC$;SPC(10);C$
70 PRINTC$TAB(10)C$
100 PRINT#4:CLOSE4:STOP
```

```
0123456789012345678901234567890123456789
DIECI CAR.          DIECI CAR.
DIECI CAR.          DIECI CAR.
```

Nei due programmi precedenti la stampa viene ottenuta trasferendo l'uscita dal video con CMD; abbiamo modificato i due esempi e stampato con PRINT#, per mostrare che TAB e SPC agiscono ancora nello stesso modo. Riportiamo i due listati INC1BIS e INC2BIS con relativi risultati.

```
1 REM INC1BIS
10 OPEN4,4
20 A$="0123456789":B$=""
30 FORK=1TO4:B$=B$+A$:NEXTK
40 PRINT#4,B$
50 PRINT#4,TAB(0)"POS0";TAB(6)"POS10";
55 PRINT#4,TAB(15)"POS30"
60 PRINT#4,TAB(80);"NUOVA RIGA"
100 PRINT#4:CLOSE4:STOP
```

```
0123456789012345678901234567890123456789
POS0      POS10               POS30

NUOVA RIGA
```

```
1 REM INC2BIS
10 OPEN4,4
20 A$="0123456789":B$=""
30 FORK=1TO4:B$=B$+A$:NEXTK
40 PRINT#4,B$
50 C$="DIECI CAR."
60 PRINT#4,C$;SPC(10);C$
70 PRINT#4,C$TAB(10)C$
100 PRINT#4:CLOSE4:STOP
```

```
0123456789012345678901234567890123456789
DIECI CAR.          DIECI CAR.
DIECI CAR.          DIECI CAR.
```

Nel programma INC3 che segue, mostriamo come incolonnare i numeri riducendoli, con l'aggiunta di spazi a sinistra, tutti con lo stesso numero di caratteri.

```
1 REM INC3
5 DIMC(7),C$(7):SP$="          "
10 OPEN4,4:CMD4
20 A$="0123456789":B$=""
30 FORK=1TO4:B$=B$+A$:NEXTK
40 PRINTB$
50 DATA1234,13567,45,890
55 DATA5432,9876,3456,12345
60 FORK=0TO7:READC(K)
63 C$(K)=STR$(C(K)):NEXTK
65 FORJ=0TO1:FORK=0TO3:PRINTC(K+J*4);"   ";
66 NEXTK:PRINT:NEXTJ
69 T$="TABELLA NUMERI IN COLONNA"
70 PRINT:PRINTT$:PRINT
75 PRINTB$
80 FORJ=0TO1:FORK=0TO3
83 PRINTRIGHT$(SP$+C$(K+J*4),10);
85 NEXTK:PRINT:NEXTJ
100 PRINT#4:CLOSE4:STOP
```

RISULTATI PROGRAMMA INC3

```
0123456789012345678901234567890123456789
 1234     13567     45     890
 5432     9876     3456     12345

TABELLA NUMERI IN COLONNA

0123456789012345678901234567890123456789
      1234     13567        45       890
      5432      9876      3456     12345
```

Nel programma INC4 invece si ottengono gli allineamenti sfruttando la funzione LEN e la funzione SPC.

```
1 REM INC4
5 DIMC(7),C$(7):SP$="            "
10 OPEN4,4:CMD4
20 A$="0123456789":B$=""
30 FORK=1TO4:B$=B$+A$:NEXTK
40 PRINTB$
50 DATA1234,8976,3456,13567,45,890,5432,9876
60 FORK=0TO7:READC(K):C$(K)=STR$(C(K)):NEXTK
63 FORJ=0TO1:FORK=0TO3:PRINTC(K+J*4);"   ";
65 NEXTK:PRINT:NEXTJ
67 T$="TABELLA NUMERI IN COLONNA"
70 PRINT:PRINTT$:PRINT:PRINTB$
80 FORJ=0TO1:FORK=0TO3
83 PRINTSPC(10-LEN(C$(K+J*4)))C$(K+J*4);
85 NEXTK:PRINT:NEXTJ
100 PRINT#4:CLOSE4:STOP
```

RISULTATI PROGRAMMA INC4

```
0123456789012345678901234567890123456789
 1234     8976     3456     13567
 45     890     5432     9876

TABELLA NUMERI IN COLONNA

0123456789012345678901234567890123456789
      1234      8976      3456     13567
        45       890      5432      9876
```

Il programa INC5 infine mostra come si possono allineare a sinistra delle parole usando le funzioni LEN e SPC.

```
1 REM INC5
5 DIMC$(11)
10 OPEN4,4:CMD4
50 DATABELLO,BENISSIMO,ALLEGRO
51 DATAPIACEVOLE,RILASSANTE,SOLEGGIATO
53 DATAGRADEVOLE,UTILE,DEFINITIVO
55 DATAARIOSO,STUPENDO,MAGNIFICO
60 FORK=0TO11:READC$(K):NEXTK
```

```
65 FORJ=0TO2:FORK=0TO3:PRINTC$(K+J*4);"   ";
67 NEXTK:PRINT:NEXTJ
70 N=0:FORK=0TO11
71 IFLEN(C$(K))>NTHENN=LEN(C$(K))
75 NEXTK:PRINT:PRINT
80 N=N+3:FORJ=0TO2:FORK=0TO3
83 PRINTC$(K+J*4);SPC(N-LEN(C$(K+J*4)));
85 NEXTK:PRINT:NEXTJ
100 PRINT#4:CLOSE4:STOP
```

RISULTATI PROGRAMMA INC5

```
BELLO   BENISSIMO   ALLEGRO   PIACEVOLE
RILASSANTE   SOLEGGIATO   GRADEVOLE   UTILE
DEFINITIVO   ARIOSO   STUPENDO   MAGNIFICO


BELLO          BENISSIMO      ALLEGRO        PIACEVOLE
RILASSANTE     SOLEGGIATO     GRADEVOLE      UTILE
DEFINITIVO     ARIOSO         STUPENDO       MAGNIFICO
```

5.6 COPIA DEL VIDEO SU CARTA

Il problema della copia su carta del video si pone nei seguenti termini:

.. Sul video i caratteri sono ottenuti con matrici di punti 8x8, in ogni linea entrano 40 caratteri e le linee sono 25. Si hanno in tutto 1000 caratteri e 64000 punti.

.. La stampante usa una matrice di punti 6x7 e quindi la rappresentazione dei caratteri non è uguale a quella visibile sul video. Ogni linea di stampa può contenere 80 caratteri e quindi di più di quanti ne servano per una linea video. Una linea di stampa può contenere fino a 480 punti.

.. La stampante riceve dal calcolatore i caratteri da stampare sotto forma di codici ASCII, va a prelevare dalla sua ROM interna la rappresentazione dei caratteri e stampa. Essa ha bisogno di ricevere gli appositi codici di controllo per cambiare set di caratteri o per stampare in campo inverso.

.. La stampante può ricevere dal calcolatore il codice per stampa grafica (8) e dopo accetta codici numerici che rappresentano una colona di punti, formata da 7 punti.

Facciamo seguire 3 diversi programmi che stampano su carta il contenuto del video, accompagnandoli con qualche commento.

Segue il programma COPIAVIDEO1; quello che vedi non è un normale listato ottenuto con LIST, ma il programma copiato da se stesso dal video (dove naturalmente è stato ottenuto con LIST). Ti accorgi facilmente di questo osservando che le linee sono rigorosamente di 40 caratteri. I listati sono due; infatti il programma copia il video due volte cambiando il set di caratteri.

```
1 REM COPIAVIDEO1
2 REM COPIA SET MAIUSCOLO/GRAFICO
3 H1$=CHR$(145):GOSUB10000
4 H1$=CHR$(17):GOSUB10000:STOP
10000 REM HARD1
10001 OPEN4,4:PRINT#4
10002 H1=256*PEEK(648)-40
10003 FORH0=0TO24:H0$=H1$:H1=H1+40
10004 FORH2=H1TOH1+39:H3=PEEK(H2)
10005 IFH3>128THENH3=H3-128:H4=1:H0$=H0$
+CHR$(18)
10006 IF(H3>0)*(H3<32)THENH3=H3+64:GOTO1
0010
10007 IF(H3>31)*(H3<64)THEN10010
10008 IF(H3>63)*(H3<96)THENH3=H3+128:GOT
O10010
10009 IF(H3>95)*(H3<128)THENH3=H3+64:GOT
O10010
10010 H0$=H0$+CHR$(H3)
10011 IFH4=1THENH0$=H0$+CHR$(146):H4=0
10012 NEXTH2:PRINT#4,H0$:NEXTH0
10013 PRINT#4:CLOSE4:RETURN
READY.
RUN
```

```
1 rem copiavideo1
2 rem copia set maiuscolo/grafico
3 h1$=chr$(145):gosub10000
4 h1$=chr$(17):gosub10000:stop
10000 rem hard1
```

```
10001 open4,4:print#4
10002 h1=256*peek(648)-40
10003 forh0=0to24:h0$=h1$:h1=h1+40
10004 forh2=h1toh1+39:h3=peek(h2)
10005 ifh3>128thenh3=h3-128:h4=1:h0$=h0$
+chr$(18)
10006 if(h3>0)*(h3<32)thenh3=h3+64:goto1
0010
10007 if(h3>31)*(h3<64)then10010
10008 if(h3>63)*(h3<96)thenh3=h3+128:got
o10010
10009 if(h3>95)*(h3<128)thenh3=h3+64:got
o10010
10010 h0$=h0$+chr$(h3)
10011 ifh4=1thenh0$=h0$+chr$(146):h4=0
10012 nexth2:print#4,h0$:nexth0
10013 print#4:close4:return
ready.
run
```

Il programma è formato da 4 linee di testata che si limitano a preparare la stringa H1$ con il codice di controllo del set maiuscolo/grafico e a chiamare il sottoprogramma HARD1, e poi a modificare il codice di controllo per il set maiuscolo/minuscolo e a richiamare ancora HARD1.
Il sottoprogramma HARD1 calcola l'indirizzo di inizio della mappa video alla linea 10002, poi con la funzione PEEK preleva i caratteri dal video (essi sono in D/CODE), controlla se sono maggiori di 128 per inviare il carattere di attivazione del campo inverso, poi li trasforma in codice ASCII. I caratteri sono ammucchiati in una stringa e poi stampati una linea video dopo l'altra. Essi vengono stampati nel set attivo sulla stampante e viene rispettato il campo inverso.

Segue il programma COPIAVIDEO2, che, come vedi, è molto semplice. Esso è formato da poche linee di testata e da un sottoprogramma. Alla linea 2 viene posto SA=0 e viene chiamato il sottoprogramma ottenendo la copia del video con il carattere standard. Alla linea 3 viene posto SA=7 e richiamato il sottoprogramma ottenendo la stampa nell'altro set. Il sottoprogramma HARD2 si limita ad aprire il video come file logico 3 e ad aprire la stampante; poi viene letto il video con 40 comandi GET#3 per riga formando una stringa di 40 caratteri che viene stampata. Alla linea 10030 si deve terminare con punto e virgola perchè il carattere "a capo" viene già letto con GET#3. Anche questa volta i listati del programma sono ottenuti con l'uso del programma stesso. Questo programma non rispetta i caratteri

in campo inverso, infatti il codice ASCII ottenuto con GET non riporta notizie sul campo. Alla linea 10005 vedi in campo inverso il carattere compreso tra le virgolette.

```
LIST

1 REM COPIAVIDEO2
2 SA=0:GOSUB10000
3 SA=7:GOSUB10000
4 STOP
10000 REM HARD2
10005 PRINT"■";
10007 OPEN3,3:OPEN4,4,SA
10010 FORK=0TO24:A$=""
10015 FORJ=0TO39
10020 GET#3,B$:A$=A$+B$
10025 NEXTJ
10030 PRINT#4,A$;
10035 NEXTK:PRINT
10040 CLOSE3:CLOSE4
10045 RETURN
READY.
RUN
```

```
1 rem copiavideo2
2 sa=0:gosub10000
3 sa=7:gosub10000
4 stop
10000 rem hard2
10005 print"■";
10007 open3,3:open4,4,sa
10010 fork=0to24:a$=""
10015 forj=0to39
10020 get#3,b$:a$=a$+b$
10025 nextj
10030 print#4,a$;
10035 nextk:print
10040 close3:close4
10045 return
ready.
run
```

Infine segue il programma COPIAVIDEO3 che copia le prime 5 linee del video ricostruendo sulla stampante per colonne di punti quello che legge sul video. Si tratta di un lavoro abbastanza complicato e risulta lento programmato in BASIC, ma è interessante come algoritmo; infatti si lavora in BASIC a livello di bit.
La testata del programma è formata da quattro sequenze in ognuna delle quali si pulisce il video, si predispone il set di caratteri e il puntatore D alla mappa dei caratteri in ROM, si chiama il sottoprogramma in 100 che stampa sul video tutti i caratteri stampabili (in tutto sono 192) occupando quasi 5 linee. I set predisposti sono i due disponibili, prima in campo diretto e poi in campo inverso.
Il sottoprogramma HARD3 lavora così:
.. Preleva i D/CODE dal video e li memorizza nella matrice X(4,39).
.. Esegue il sottoprogramma in 20000, che, dopo aver reso disponibile la ROM dei caratteri (linee 20000 e 20001) va a plelevare per ogni carattere gli 8 byte che lo descrivono e li memorizza incolonnandoli nella matrice D(41,39). Dopo rilascia la ROM (linee 20020 e 20021) e termina.
.. Esegue il sottoprogramma in 20035 che preleva dalla matrice D(41,39) a gruppi di 7 le linee ricostruendo le colonne di punti da inviare come caratteri grafici alla stampante. Viene formata una stringa per ogni gruppo di 7 righe della matrice e viene stampata. Per calcolare i codici dei caratteri grafici si usa la matrice Z(39,7). Naturalmente la prima stringa stampata non rappresenta la prima linea del video, infatti ha lavorato su 7 righe della matrice e non su 8, ma con la seconda stampa si completano i caratteri e comincia ad essere stampata la seconda linea del video. Con 7 stampe si termina. La matrice D è stata dimensionata a 42 righe anche se ne vengono riempite solo 40 (5x8=40) per avere un multiplo di 7 e poter completare la stampa.

```
1 REM COPIAVIDEO3
2 REM COPIA SET CARATTERI 5 RIGHE
3 DIMX(4,39),D(41,39),Z(39,7)
50 H2$=CHR$(142)+CHR$(146)
55 GOSUB100:D=0:GOSUB10000
57 H2$=CHR$(142)+CHR$(18)
59 GOSUB100:D=0:GOSUB10000
70 H2$=CHR$(14)+CHR$(146)
75 GOSUB100:D=1:GOSUB10000
77 H2$=CHR$(14)+CHR$(18)
79 GOSUB100:D=1:GOSUB10000
81 PRINTCHR$(146)CHR$(142):STOP
100 PRINT"♥";H2$;
105 FORK=32TO127:PRINTCHR$(K);:NEXTK
110 FORK=160TO255:PRINTCHR$(K);:NEXTK
115 PRINT:RETURN
```

```
10000 REM HARD3
10005 OPEN4,4
10010 H1=256*PEEK(648)-40
10015 FORH0=0TO4:H1=H1+40
10020 J=0:FORH2=H1TOH1+39
10021 X(H0,J)=PEEK(H2)
10060 J=J+1:NEXTH2:NEXTH0
10063 GOSUB20000
10064 GOSUB20035
10065 PRINT#4:CLOSE4:RETURN
20000 POKE56334,PEEK(56334)AND254
20001 POKE1,PEEK(1)AND251
20005 A=53248+D*2048
20007 FORM=0TO4:FORI=0TO39:Y=X(M,I)
20010 FORK=0TO7:D(M*8+K,I)=PEEK(A+Y*8+K)
20011 NEXTK:NEXTI:NEXTM
20020 POKE1,PEEK(1)OR4
20021 POKE56334,PEEK(56334)OR1:RETURN
20035 FORN=0TO41STEP7
20036 FORI=0TO39:FORL=0TO7:Z(I,L)=0
20037 NEXTL:NEXTI
20038 FORI=0TO39:FORM=0TO6:Y=D(M+N,I)
20043 FORL=0TO7
20045 IFINT(Y/2↑(7-L))=0THEN20060
20050 Z(I,L)=Z(I,L)+2↑M:Y=Y-2↑(7-L)
20060 NEXTL:NEXTM:NEXTI:PRINT#4,CHR$(8);
20065 A$="":FORI=0TO39:FORL=0TO7
20066 A$=A$+CHR$(Z(I,L)+128)
20067 IFI=19THENPRINT#4,A$;:A$=""
20070 NEXTL:NEXTI:PRINT#4,A$
20080 NEXTN:PRINT#4,CHR$(15):RETURN
```

RISULTATI DEL PROGRAMMA COPIAVIDEO3

Come vedi i caratteri ottenuti sono un pò diversi da quelli di tutti gli altri listati di questo libro, sono un pò più grassottelli.

Questo algoritmo programmato in linguaggio macchina sarebbe molto più veloce.

CAPITOLO 6

COSTRUZIONE DI PROGRAMMI

6.1 INTRODUZIONE

Lo scopo che ci siamo proposti di raggiungere con la stesura di questo libro è sempre stato quello di insegnare a scrivere programmi in Basic per il calcolatore COMMODORE 64. I numerosissimi esempi sempre riportati lo testimoniano continuamente. Tuttavia ogni esempio si riferisce ad uno o più casi specifici e sicuramente, soprattutto se sei un principiante, avrai delle difficoltà a "pensare come costruire un programma" che serva a risolvere un problema.

Nel Paragrafo 3 del Capitolo 1 abbiamo dato una specie di schema di quello che si deve fare per impostare lo studio di un lavoro da programmare per il calcolatore. A questo punto, ti può forse essere utile tornare su quelle pagine. Quello che ti raccomandiamo nuovamente è di non metterti mai a programmare con idee vaghe in testa direttamente sul calcolatore.
Una cosa che noi riteniamo molto utile è l'esame attento e critico di programmi già funzionanti e ben documentati. In questo Capitolo sono riportati tre esempi di costruzione di programmi, ci auguriamo tu possa trovarli di tuo interesse.

Una buona tecnica da usare per affrontare un problema è il METODO TOP DOWN, cioè dall'alto al basso. Esso consiste nel fare inizialmente uno schema molto generale del lavoro e poi successivamente sviluppare ogni parte affinando sempre di più l'analisi specifica. L'ultimo affinamento sono le istruzioni del programma. Per applicare questo metodo ci si può servire sia dei diagrammi a blocchi che della descrizione verbale.

Prima di procedere vogliamo spendere alcune frasi a proposito della DOCUMENTAZIONE, diciamo, ESTERNA dei programmi. La DOCUMENTAZIONE alla quale abbiamo fatto riferimento fino ad ora è quella INTERNA che deve essere di aiuto nello stendere il programma e che risulta ASSOLUTAMENTE NECESSARIA se, dopo un pò di tempo, si devono apportare modifiche al programma. E' incredibile come si dimentica facilmente quello a cui ci si è dedicati con tanta attenzione!

Per documentazione esterna intendiamo il manuale operativo che deve accompagnare ogni programma che venga prodotto per essere utilizzato da altre persone (e anche dagli autori). Il compilatore PETSPEED è accompagnato dal suo manuale operativo e così è per tutto il software che si compra. Tuttavia, ogni acquirente vorrebbe che il manuale contenesse una maggior quantità di informazioni. E' un problema molto spinoso e ben noto a chi lavora nel campo dell'informatica.

6.2 ALGORITMI DI ORDINAMENTO

Ci siamo proposti di confrontare tra loro tre algoritmi di ordinamento memorizzando i tempi di esecuzione.
Abbiamo deciso di creare un vettore di numeri interi positivi minori o uguali a 9999 estraendo N numeri a caso. Per poter provare il programma anche in tempi successivi lavorando con gli stessi numeri, abbiamo eseguito la prima estrazione di numero a caso con argomento negativo e le successive con argomento positivo, non importa quale.
Per lasciare generalità al programma, all'inizio viene chiesto il numero negativo da usare come argomento per la prima estrazione, e il numero N di numeri sui quali si vuole lavorare.
La fase iniziale del programma consiste nell'estrarre N numeri e memorizzarli in un vettore di N elementi. Dato che i metodi di ordinamento lavorano sullo stesso vettore che contiene i numeri, abbiamo dovuto usare due vettori, uno per ordinare e l'altro per mantenere i numeri nello stesso ordine iniziale, altrimenti non erano possibili i confronti sui tempi di esecuzione. Ogni volta, prima di chiamare il sottoprogramma di ordinamento si trasferiscono i numeri originali nel vettore dove vengono poi ordinati.
Gli ordinamenti vengono fatti in senso crescente.

Passiamo a descrivere la sequenza delle operazioni:

a) preparazione di alcune stringhe per intestazione risultati;
b) richiesta della base negativa S per l'estrazione e del numero N dei numeri da estrarre, con controllo di validità;
c) apertura della stampante e stampa intestazione risultati;
d) richiesta per decidere se stampare tutti i numeri o solo i tempi di esecuzione;
e) dimensionamento dei due vettori N% e P% per gli N numeri e definizione della funzione FNY per rendere i numeri estratti compresi tra 0 e 9999;
f) estrazione dei numeri a caso con controllo del tempo;
g) preparazione dei titoli e chiamata del sottoprogramma di stampa;
f) preparazione dei titoli, chiamata del sottoprogramma di ordinamento con metodo 1, chiamata del sottoprogramma di stampa;
h) preparazione dei titoli, chiamata del sottoprogramma di ordinamento con metodo 2, chiamata del sottoprogramma di stampa;

i) preparazione dei titoli, chiamata del sottoprogramma di ordinamento con metodo 3, chiamata del sottoprogramma di stampa;
l) stop del programma.

Il sottoprogramma di stampa opera così:

m) stampa un titolo, se non si vuole la stampa dei numeri va in o);
n) stampa la tabella dei numeri 5 per riga;
o) stampa il tempo che è stato precedentemente memorizzato leggendo il valore iniziale e finale del clock.

Il metodo 1, chiamato "ordinamento a bolle"; opera così:

a1) confronta i numeri a coppie, primo con secondo, secondo con terzo, fino all'ultima coppia, penultimo con ultimo; se dal confronto risulta che qualche coppia non è in ordine viene fatto lo scambio dei numeri e viene modificato un indicatore a ricordare che si è operato uno scambio;
a2) se alla fine del ciclo l'indicatore (azzerato a inizio ciclo) è rimasto a zero, significa che la tabella è stata trovata in ordine e si esce;
a3) se invece l'indicatore è stato modificato, si ricominciano i confronti dall'inizio, ma si tralascia un elemento in coda; infatti ad ogni ciclo l'elemento maggiore scende all'ultimo posto disponibile.

Tale metodo opera un numero di confronti variabile, che dipende dal disordine dei numeri.

Il metodo 2 chiamato, "ordinamento con dimezzamento dell'intervallo" assomiglia al precedente, ma invece di confrontare due elementi che si trovano vicini (intervallo 1), confronta due elementi che distano dapprima N/2, poi N/4, e così via fino ad arrivare alla distanza 1. Alla fine vengono percorsi un numero variabile di cicli con intervallo 1, fino al raggiunto ordine.
Anche qui il numero dei cicli è variabile e dipende dal disordine iniziale dei numeri.

Il metodo 3, chiamato "ordinamento a cicli fissi", opera un numero fisso di cicli di confronto. Se gli elementi sono N, esso percorre N-1 cicli, in ognuno dei quali confronta il primo elemento disponibile con tutti gli altri e memorizza l'indice dell'elemento che risulta minore e che diventa il nuovo termine di paragone. Alla fine del ciclo il termine di paragone è l'elemento minore ed esso viene spostato nella posizione del primo elemento disponibile mediante un doppio scambio tra i due elementi interessati. Al ciclo successivo si parte da una posizione più avanti; l'ultimo ciclo è il confronto tra due elementi.
Questo metodo fa sempre lo stesso numero di confronti anche se inizia a lavorare su una tabella ordinata.

Riportiamo i diagrammi a blocchi del programma principale, del sottoprogramma di stampa e dei tre sottoprogrammi di ordinamento.

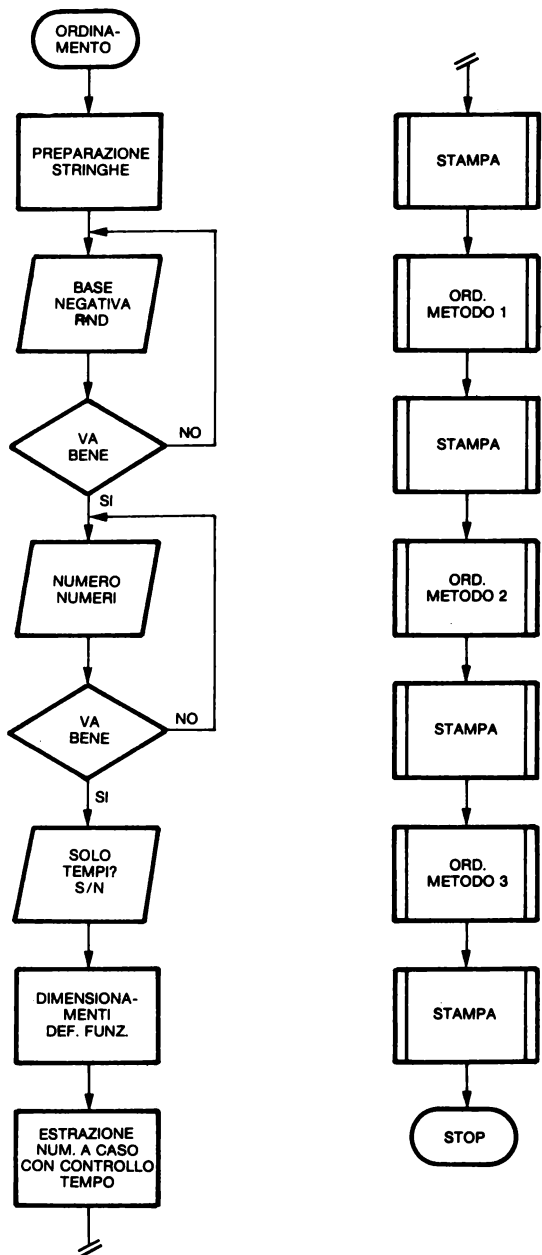



Figura 6.1 Diagramma a blocchi ORDINAMENTO

SOTTOPROGRAMMA 100

APERTURA STAMPANTE

TITOLI TABELLA

SOLO TEMPI

SI

NO

CONT. ELEM. RIGA: IR = 0

INIZIO CICLO DATI JR = 0

STAMPA DATO

IR = IR + 1

GIA' 5 ELEM.

SI

A CAPO

IR = 0

JR = JR + 1

JR > N–1

NO

SI

STAMPA TEMPI

CHIUSURA STAMPANTE

RETURN

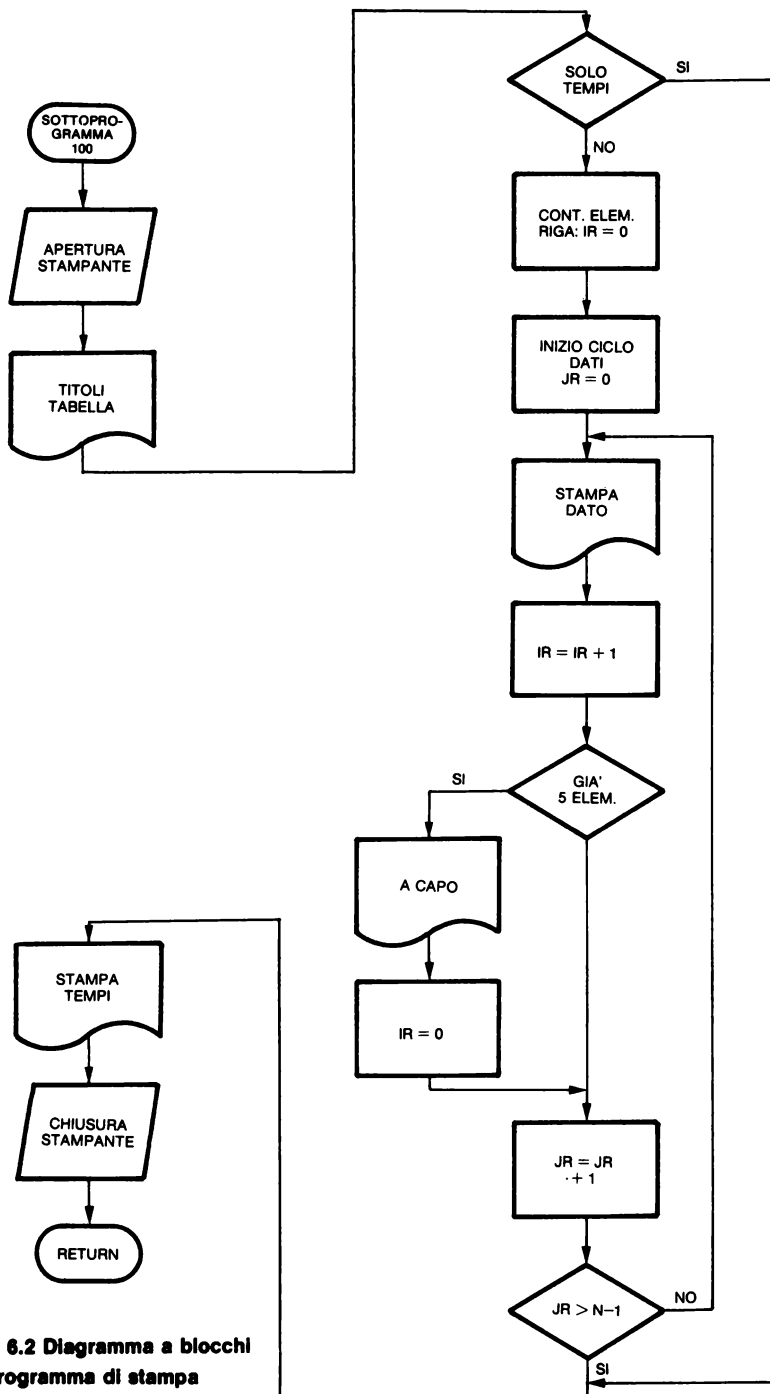



Figura 6.2 Diagramma a blocchi sottoprogramma di stampa

SOTTOPRO-
GRAMMA
200

L = N-2
MEM. CLOCK

INDICATORE
J = 0

INIZIO CICLO
CONFRONTI
K=0 L

N% (K)
≤
N% (K+1)

SI

NO

SCAMBIO
ELEMENTI
J=1

FINE
CICLO

NO

SI

L = L−1

NO

J = 0

SI

MEM. CLOCK

RETURN

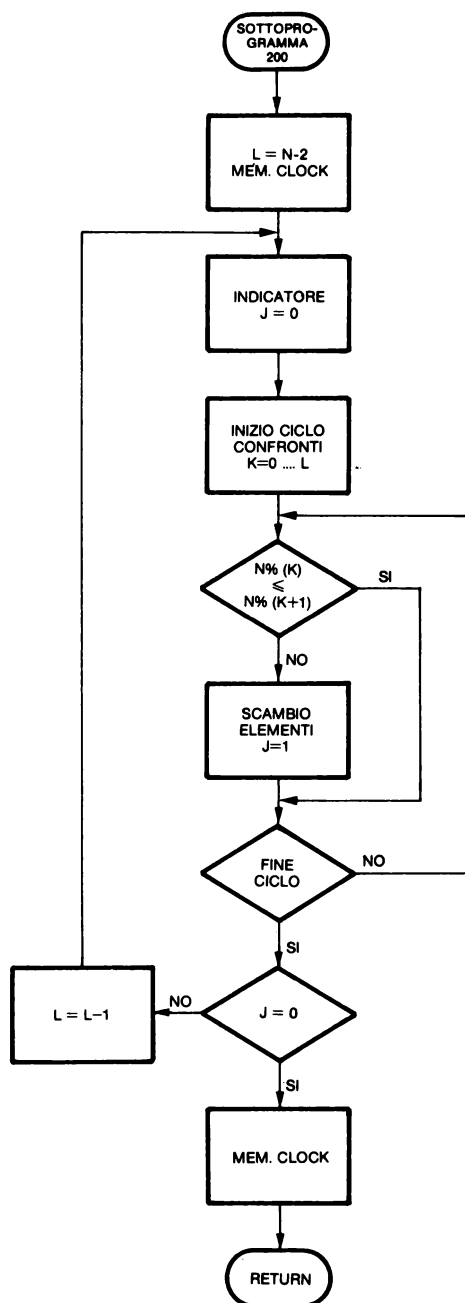



Figura 6.3 Diagramma a blocchi metodo 1 (linea 200)

SOTTOPRO-
GRAMMA
300

IN = N/2
MEM. CLOCK

INDICATORE
J=0

INIZIO CICLO
N=0 .... (N−1−IN)

N% (K)
≤ N% (K + IN)

SI

NO

SCAMBIO
ELEMENTI
J=1

FINITO
CICLO

NO

IN = IN/2

SI

NO

SI

IN = 1

NO

IN=1
E J = 0

SI

MEM. CLOCK

RETURN

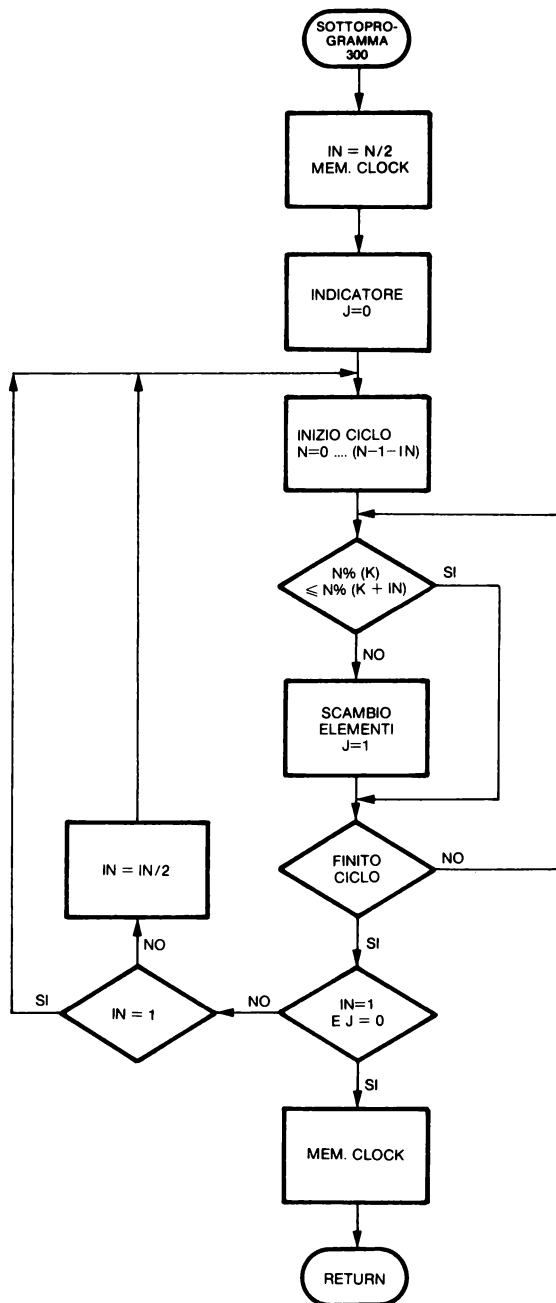



Figura 6.4 Diagramma a blocchi metodo 2 (linea 300)

SOTTOPRO-
GRAMMA
400

K=0
MEM. CLOCK

J=K+1
I = K

INIZIO CICLO
L=J ... (N-1)

N% (K)
≤ N% (L)

SI

NO

K = L

NO

FINE
CICLO

SI

K = I

SI

NO

SCAMBIO
ELEMENTI

K = I+1

MEM. CLOCK

RETURN

K = N−1

NO

SI

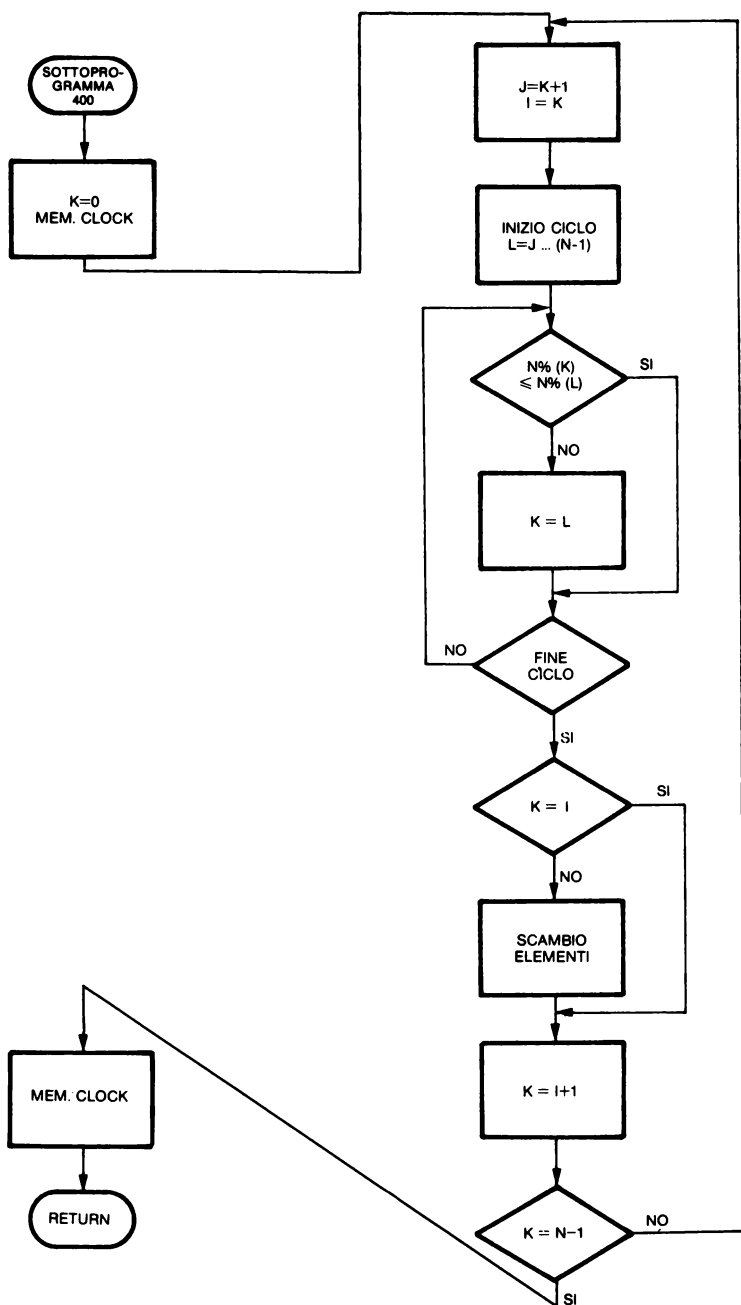



Figura 6.5 Diagramma a blocchi metodo 3 (linea 400)

Segue il listato del programma in Basic.

```
1 REM ORDINAMENTO
5 REM GENERAZIONE N NUMERI A CASO
7 REM PRIMO RND ARGOMENTO NEGATIVO
9 S$="            ":X$="ORDINATI CON METODO "
11 Y$="ORDINAMENTO"
12 INPUT"J■■BASE NEGATIVA RND: ";S
13 IFS>=0THEN12
14 INPUT"■QUANTI NUMERI: ";N
15 IFN<=0THEN14
17 OPEN4,4:CMD4:PRINT"BASE RND: ";S
19 PRINT"ORDINAMENTO DI ";N;" NUMERI":PRINT
20 PRINT#4:CLOSE4
21 REM NUMERI MINORI UGUALI A 9999
23 INPUT"STAMPA SOLO TEMPI (S/N): ";R$
25 DIM N%(1000),P%(1000)
30 DEFFNY(X)=X*(9999)+1
31 A=TI
35 N%(0)=FNY(RND(S)):P%(0)=N%(0)
40 FORK=1TON-1
45 N%(K)=FNY(RND(1)):P%(K)=N%(K)
50 NEXTK:B=TI
55 A$="ESTRATTI":B$="ESTRAZIONE":GOSUB100
60 C$=" 1":A$=X$+C$:B$=Y$
63 GOSUB200:GOSUB100
65 C$=" 2":A$=X$+C$:GOSUB500
67 GOSUB400:GOSUB100
69 C$=" 3":A$=X$+C$:GOSUB500
71 GOSUB300:GOSUB100
80 PRINT"BYTE LIBERI: ";FRE(0)
99 STOP
100 OPEN4,4:CMD4
105 PRINT"TABELLA NUMERI "A$:PRINT
107 IFR$="S"THEN125
110 IR=0:FORJR=0TON-1
115 N$=STR$(N%(JR)):N$=LEFT$(N$+S$,8)
117 PRINTN$;:IR=IR+1
120 IFIR=5THENPRINT:IR=0
123 NEXTJR:PRINT
125 PRINT"TEMPO "B$;(B-A)/60;" SEC."
130 PRINT#4:CLOSE4:RETURN
```

```
200 REM METODO A BOLLE
205 L=N-2:A=TI
210 J=0:FORK=0TOL
215 IFN%(K)<=N%(K+1)THEN225
220 C%=N%(K):N%(K)=N%(K+1):N%(K+1)=C%:J=1
225 NEXTK
230 IFJ=0THENB=TI:RETURN
235 L=L-1:GOTO210
300 REM METODO DIMEZZAMENTO INTERVALLO
305 IN=INT(N/2+0.5):A=TI
310 J=0:FORK=0TO(N-1-IN)
315 IFN%(K)<=N%(K+IN)THEN320
317 C%=N%(K):N%(K)=N%(K+IN):N%(K+IN)=C%:J=1
320 NEXTK
325 IFIN=1ANDJ=0THENB=TI:RETURN
327 IFIN=1THEN310
330 IN=INT(IN/2+0.5):GOTO310
400 REM METODO CICLI FISSI
405 K=0:A=TI
410 J=K+1:I=K
415 FORL=JTON-1
420 IFN%(K)<=N%(L)THEN430
425 K=L
430 NEXTL
435 IFK=ITHEN445
440 C%=N%(I):N%(I)=N%(K):N%(K)=C%
445 K=I+1:IFK=N-1THENB=TI:RETURN
450 GOTO410
500 FORK=0TON-1:N%(K)=P%(K):NEXTK:RETURN
```

VARIABILI USATE NEL PROGRAMMA

S$	stringa di 8 spazi
X$	stringa descrittiva
Y$	" "
R$	stringa per risposta s/n
A$	stringa descrittiva
B$	" "
C$	" "
N$	stringa per conversione numeri da allineare
S	base negativa estrazione numeri a caso
N	numero dei numeri su cui lavorare

N%(N-1)	vettore di numeri interi per i numeri estratti e da ordinare
P%(N-1)	vettore di numeri interi per conservare i numeri estratti
X	variabile di lavoro
A	variabile per tempo iniziale di una operazione
B	variabile per tempo finale di una operazione
K	variabile di controllo ciclo
IR	variabile per contare i numeri stampati su una riga
JR	variabile controllo ciclo
L	puntatore per ordinamento
J	indicatore per ordinamento
C%	variabile di comodo per operare doppio scambio
IN	variabile di lavoro, intervallo confronto

Riportiamo i risultati di tre prove; per i 57 numeri riportiamo anche i listati dei numeri, per le altre due solo i tempi di esecuzione.

```
BASE RND: -5890
ORDINAMENTO DI  57  NUMERI


TABELLA NUMERI ESTRATTI

 1        9386     2926     697      6505
 2284     2676     7538     5450     4055
 1595     5109     4185     5644     8507
 5934     3863     691      8831     1688
 6656     8902     2587     4152     2840
 9343     7        6788     9839     1303
 4980     9353     8985     855      1793
 4750     1706     5087     4682     1796
 8806     5604     7490     531      6088
 7991     448      2212     660      4223
 1776     2133     5284     1088     3784
 4158     3662
TEMPO ESTRAZIONE 1.7333333  SEC.
```

```
TABELLA NUMERI ORDINATI CON METODO  1

 1          7          448        531        660
 691        697        855        1088       1303
 1595       1688       1706       1776       1793
 1796       2133       2212       2284       2587
 2676       2840       2926       3662       3784
 3863       4055       4152       4158       4185
 4223       4682       4750       4980       5087
 5109       5284       5450       5604       5644
 5934       6088       6505       6656       6788
 7490       7538       7991       8507       8806
 8831       8902       8985       9343       9353
 9386       9839
TEMPO ORDINAMENTO 37.5666667  SEC.

TABELLA NUMERI ORDINATI CON METODO  2

 1          7          448        531        660
 691        697        855        1088       1303
 1595       1688       1706       1776       1793
 1796       2133       2212       2284       2587
 2676       2840       2926       3662       3784
 3863       4055       4152       4158       4185
 4223       4682       4750       4980       5087
 5109       5284       5450       5604       5644
 5934       6088       6505       6656       6788
 7490       7538       7991       8507       8806
 8831       8902       8985       9343       9353
 9386       9839
TEMPO ORDINAMENTO 20.0333333  SEC.
```

```
TABELLA NUMERI ORDINATI CON METODO  3

 1          7          448        531        660
 691        697        855        1088       1303
 1595       1688       1706       1776       1793
 1796       2133       2212       2284       2587
 2676       2840       2926       3662       3784
 3863       4055       4152       4158       4185
 4223       4682       4750       4980       5087
 5109       5284       5450       5604       5644
 5934       6088       6505       6656       6788
 7490       7538       7991       8507       8806
 8831       8902       8985       9343       9353
 9386       9839
TEMPO ORDINAMENTO 17  SEC.


BASE RND: -34
ORDINAMENTO DI  250  NUMERI


TABELLA NUMERI ESTRATTI

TEMPO ESTRAZIONE 7.08333334  SEC.

TABELLA NUMERI ORDINATI CON METODO  1

TEMPO ORDINAMENTO 698.8  SEC.

TABELLA NUMERI ORDINATI CON METODO  2

TEMPO ORDINAMENTO 339.166667  SEC.

TABELLA NUMERI ORDINATI CON METODO  3

TEMPO ORDINAMENTO 271.166667  SEC.
```

```
BASE RND: -3
ORDINAMENTO DI  503  NUMERI


TABELLA NUMERI ESTRATTI

TEMPO ESTRAZIONE 14.2  SEC.

TABELLA NUMERI ORDINATI CON METODO  1

TEMPO ORDINAMENTO 2904.51667  SEC.

TABELLA NUMERI ORDINATI CON METODO  2

TEMPO ORDINAMENTO 1370.38333  SEC.

TABELLA NUMERI ORDINATI CON METODO  3

TEMPO ORDINAMENTO 1593.51667  SEC.
```

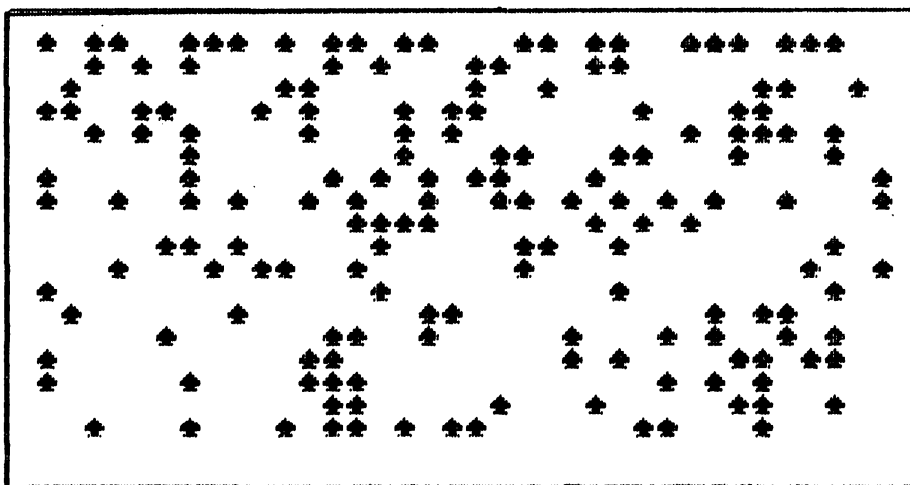
Come puoi vedere dai risultati sembra in generale più veloce il metodo 3 e più lento il metodo 1. E' evidente che se si opera con il metodo 1 su una serie di numeri già in ordine si ottiene il tempo minore.
Nell'Appendice B puoi vedere come cambiano i tempi se si compila il programma.

6.3 CARATTERI A CASO E COPIA SU CARTA

Abbiamo pensato di completare il programma COPIAVIDEO3 presentato nel Capitolo 5, portandolo a ricopiare su carta tutto il video e non solo 5 righe.
Per ottenere rapidamente qualcosa da ricopiare abbiamo impostato un programma che traccia una riquadratura sul video e, all'interno di questa, posiziona a caso un carattere scelto dall'utente. All'inizio viene chiesto quante volte si vuole posizionare a caso il carattere e quale carattere si sceglie. Questo non è un programma di grafica ad alta risoluzione, infatti l'argomento verrà trattato nell'apposito volume. Il video viene riempito usando i caratteri normali del sistema e non quelli che può creare l'utente.
Il programma TUTTOVIDEO con HARD5, resta però valido anche se il video è riempito con grafica ad alta risoluzione, infatti esso lavora per punti e non per

caratteri, anzi per colonne di punti (7 per volta).

Per far girare il sottoprogramma per tutto il video abbiamo dovuto fare un pò di conti: il video contiene in tutto 1000 caratteri e nella MAPPA VIDEO si trovano 1000 codici che fanno da puntatori alla descrizione dei caratteri (sia essa in ROM o in RAM). Per poter sviluppare la descrizione di ogni carattere in 8 byte occorrono 1000x8=8000 byte. Il Basic non ci consente di indirizzare i singoli byte, però se usiamo variabili con indice intere, esso usa 2 byte per ogni elemento, quindi per i nostri 8000 byte descrittori ne impegnamo 16000 sotto forma di variabili intere con indice. Con artifici di programmazione si potrebbe mettere in ogni variabile intera con indice 2 byte descrittori, infatti ognuno di essi è minore di 255, ma non vogliamo complicarci troppo la vita.
Risulta chiaro che un buon programma di HARD COPY deve essere scritto in linguaggio macchina. Noi con la scusa della copia del video riusciamo ad impostare discorsi interessanti ai fini dell'apprendimento della programmazione in Basic e questo è il nostro scopo attuale.
A questo punto non è stato difficile modificare il programma COPIAVIDEO3. Abbiamo dimensionato le 3 matrici di variabili intere:
.. X%(24,39) per contenere la MAPPA VIDEO;
.. D%(202,39) per contenere la descrizione in 8 byte di ogni carattere del video, 25x8=200, ma abbiamo dovuto aggiungere 3 per avere un multiplo di 7;
.. Z(39,7) per sviluppare le colonne di punti, 8 per ogni posizione video.
Alla linea 10015 il ciclo va da 0 a 24; alla linea 20007 il ciclo va da 0 a 24; alla linea 20035 il ciclo va da 0 a 202.Per il resto nessuna modifica.

Segue un risultato del programma:

Segue il listato del programma DISE1.

```
1 REM DISE1
2 PRINT"■■QUANTE ESTRAZIONI: ";:INPUTZ
3 IFZ<0THEN2
5 V$="■■■■■■■■■■■■■■■■■■■■■■■■■"
7 PRINT"■■■QUALE CARATTERE: ";:INPUTA$
9 IFLEN(A$)<>1THEN7
10 PRINT"■■";:FORL=1TO38:PRINT"_";:NEXTL:PRINT
11 PRINTLEFT$(V$,23)"■■";
13 FORL=1TO38:PRINT"¯";:NEXTL
15 PRINT"■":FORL=1TO21:PRINT" |":NEXTL
17 PRINT"■":FORL=1TO21:PRINTTAB(38)" |":NEXTL
20 FORK=1TOZ
23 X1=RND(3):Y1=RND(3)
27 X1=INT(X1*37+1):IFX1<2THENX1=2
29 Y1=INT(Y1*20+1):IFY1<3THENY1=3
31 PRINTLEFT$(V$,Y1);TAB(X1);A$:PRINT"■"
37 NEXTK
500 REM TUTTOVIDEO3
505 DIMX%(24,39),D%(202,39),Z%(39,7)
510 H1$=CHR$(145):H2$=CHR$(142)+CHR$(146)
515 D=0:GOSUB10000
525 STOP
10000 REM HARD5
10005 OPEN4,4:PRINT#4,H1$;
10010 H1=256*PEEK(648)-40
10015 FORH0=0TO24:H1=H1+40
10020 J=0:FORH2=H1TOH1+39
10021 X%(H0,J)=PEEK(H2)
10060 J=J+1:NEXTH2:NEXTH0
10063 GOSUB20000
10064 GOSUB20035
10065 PRINT#4:CLOSE4:RETURN
20000 POKE56334,PEEK(56334)AND254
20001 POKE1,PEEK(1)AND251
20005 A=53248+D*2048
20007 FORM=0TO24:FORI=0TO39:Y%=X%(M,I)
20010 FORK=0TO7:D%(M*8+K,I)=PEEK(A+Y%*8+K)
20011 NEXTK:NEXTI:NEXTM
20020 POKE1,PEEK(1)OR4
20021 POKE56334,PEEK(56334)OR1:RETURN
```

```
20035 FORN=0TO202STEP7
20036 FORI=0TO39:FORL=0TO7:Z%(I,L)=0
20037 NEXTL:NEXTI
20038 FORI=0TO39:FORM=0TO6:Y%=D%(M+N,I)
20043 FORL=0TO7
20045 IFINT(Y%/2↑(7-L))=0THEN20060
20050 Z%(I,L)=Z%(I,L)+2↑M:Y%=Y%-2↑(7-L)
20060 NEXTL:NEXTM:NEXTI:PRINT#4,CHR$(8);
20065 A$="":FORI=0TO39:FORL=0TO7
20066 A$=A$+CHR$(Z%(I,L)+128)
20067 IFI=19THENPRINT#4,A$;:A$=""
20070 NEXTL:NEXTI:PRINT#4,A$
20080 NEXTN:PRINT#4,CHR$(15)H2$:RETURN
```

Per ottenere questo programma abbiamo lavorato così:
.. abbiamo scritto ex novo le linee da 1 a 37;
.. abbiamo verificato che il quadro video uscisse bene e abbiamo memorizzato il programma su disco;
.. abbiamo richiamato in memoria il vecchio programma COPIAVIDEO3, l'abbiamo modificato mettendo in pratica quanto detto sopra e l'abbiamo memorizzato su disco con un nome provvisorio;
.. abbiamo scritto NEW e poi abbiamo caricato in memoria il programma del quadro video già provato e corretto;
.. abbiamo letto con PEEK il contenuto dei due byte 45 e 46 e calcolato l'indirizzo di inizio delle variabili; questo indirizzo - 2 dà l'indirizzo del LINK finale del programma, quello con due zeri;
.. abbiamo scritto con POKE nei 2 byte 43 e 44 spostando l'indirizzo di inizio del programma Basic a quello calcolato dell'ultimo LINK;
.. abbiamo operato il LOAD del programma di copia video precedentemente memorizzato;
.. abbiamo scritto con POKE nei 2 byte 43 e 44 l'indirizzo solito di inizio Basic;
.. abbiamo listato il programma, l'abbiamo memorizzato su disco e DOPO provato.

Nota il DOPO; ti raccomandiamo di memorizzare sempre i programmi prima di provarli, non si sa mai quello che può succedere!

Commento al programma per linee:

.. da 1 a 2 richiesta del numero di estrazioni Z, se Z negativo viene rifatta la richiesta;
.. in 5 definizione della stringa V$; essa inizia con il carattere HOME e continua con 24 caratteri CRSR/giù, serve per muoversi in verticale sul video, usandola come

argomento della funzione LEFT$ e prendendo la parte che serve;
.. da 7 a 9 richiesta del carattere, con controllo che sia uno solo;
.. in 10 traccia il bordo superiore;
.. da 11 a 13 traccia il bordo inferiore;
.. in 15 traccia il bordo sinistro;
.. in 17 traccia il bordo destro;
.. da 20 a 37 ciclo di estrazione di Z coppie di numeri a caso con argomento di RND positivo (sequenze sempre diverse); aggiustamento di X1, coordinata orizzontale per farla muovere entro i bordi laterali; aggiustamento di Y1, coordinata verticale, per farla muovere all'interno dei bordi orizzontali; disegno del carattere nella posizione definita;
.. in 505 dimensionamento delle matrici per il sottoprogramma HARD5;
.. in 510 preparazione delle stringhe con i caratteri di controllo per la stampante e il calcolatore;
.. in 515 pone D=0 per accedere al primo set di caratteri nella ROM del calcolatore, lancia il sottoprogramma HARD5;
.. in 525 fine del programma;
.. in 10005 apre la stampante come file 4 e predispone il set normale, anche se non servirebbe, dato che si stampa in grafica;
.. in 10010 prepara H1 puntatore alla mappa video (di norma in 1024);
.. da 10015 a 10060 trasferimento della mappa video nella matrice X%; si poteva anche lavorare prelevando ogni volta dalla mappa video, ma è più comodo lavorare con le variabili con indice;
.. in 10063 chiamata del sottoprogramma 20000 che va a leggere in ROM le descrizioni dei caratteri;
.. in 10064 chiamata del sottoprogramma 20035 che stampa il video;
.. in 10065 chiusura della stampante e ritorno al programma principale;
.. da 20000 a 20001 disabilitazione I/O e interrupt per rendere disponibile la ROM dei caratteri;
.. in 20005 calcolo in base al valore di D dell'indirizzo della mappa caratteri in ROM;
.. da 20007 a 20011 prelevamento della descrizione dei caratteri e memorizzazione nella matrice D%;
.. da 20020 a 20021 disabilitazione ROM caratteri e RETURN;
.. in 20035 istituzione del ciclo FOR che gestisce le righe della matrice D% a gruppi di 7;
.. da 20036 a 20037 azzeramento della matrice Z% dove si fanno le somme dei codici per le 7 righe di punti di ogni colonna di carattere grafico;
.. in 20038 istituzione del ciclo di riga e del ciclo di colonna per la striscia di 7 linee di punti e trasferimento del byte da calcolare;
.. in 20043 istituzione del ciclo che spezza il byte in bit e gli attribuisce i giusti pesi per costruire i caratteri grafici;
.. da 20045 a 20050 calcolo;

.. in 20060 chiusura dei cicli aperti e stampa codice controllo per la grafica;
.. da 20065 a 20070 stampa dei caratteri grafici contenuti nella matrice Z% che si riferisce a una riga video; i caratteri sono 40x8=320 e quindi non possono essere ammucchiati tutti in una stringa, da cui l'analisi sul valore 19 di I per stampare il primo pezzo, poi annullare A$ e continuare;
.. in 20080 chiusura del ciclo iniziato in 20035, stampa dei codici di controllo per tornare a stampa normale e RETURN.

Il difetto di questo programma DISE1 è la lentezza; impiega circa un'ora per fare la copia del video. Abbiamo provato a compilare il programma, dopo aver apportato una modifica alla linea 10010 per togliere il riferimento ai puntatori in pagina zero (abbiamo posto l'indirizzo d'inizio della mappa video come costante). Il programma origine del compilato è memorizzato come DISE2 sul dischetto dei programmi del libro e quello compilato come DISE2.WOW. Il programma compilato impiega circa 10 minuti per fare la copia del video.

6.4 STAMPA DI UNA FATTURA PROFESSIONALE

Ci siamo proposti di scrivere un programma che consenta di stampare una fattura professionale. Una cosa molto semplice senza registrazione sui libri contabili; per quest'ultimo tipo di lavoro ti consigliamo di imparare a usare programmi della famiglia VISICALC, ne esistono varie versioni anche per il COMMODORE 64 e sono molto comodi.

Il programma deve produrre i seguenti risultati:
.. stampare il numero di copie della fattura richiesto;
.. intestare il foglio di carta con: titolo, nome e cognome, di chi emette la fattura, scrivendo in centro del foglio, in alto, a caratteri allargati;
.. porre in basso su due righe l'indirizzo del mittente e il telefono, sempre centrando la scritta e ingrandendo i caratteri;
.. porre a sinistra in alto: titolo, ragione sociale o nome e indirizzo del destinatario;
.. porre a destra il numero della fattura, la città e la data;
.. scrivere su al massimo 10 righe la causale della fattura;
.. scrivere l'importo della fattura;
.. calcolare e scrivere l'importo dell'IVA;
.. calcolare il totale e scriverlo;
.. riportare i dati anagrafici richiesti.

Per semplificare il problema abbiamo deciso che la fattura si riferisce ad un solo importo, e inoltre la causale della fattura deve essere fornita con le righe già composte per la stampa.

I dati di Input sono:
.. Il numero di copie che vuoi stampare, memorizzato in NF.
.. DATI MITTENTE: Titolo, Nome, Cognome, CAP, Città, Indirizzo, Telefono, Codice Fiscale, Partita IVA, Luogo e data di nascita. Per questi 10 dati abbiamo dimensionato il vettore M$(10) e per le relative descrizioni, da usare nella richiesta dei dati, il vettore C$(10).
.. DATI FATTURA: Numero fattura, Luogo emissione, Data fattura, per i quali abbiamo usato le variabili singole: CF$, DF$ e N$.
.. DATI DESTINATARIO: Titolo, Ragione Sociale, Indirizzo, CAP/Città. Per questi dati abbiamo dimensionato il vettore I$(4) e per le relative descrizioni il vettore CI$(4).
.. Descrizione causale fattura: per queste linee che possono essere 10 al massimo abbiamo usate il vettore Z$(10), con dimensionamento implicito.
.. Importo fattura, memorizzato in M, come numero e in M$ come stringa da stampare.
.. Percentuale IVA, memorizzata in IV.

I dati di Output sono tutti quelli ricevuti come Input, più il calcolo dell'importo IVA, memorizzato in IV$, e il totale memorizzato in FT$.

Il programma è composto da due parti:
.. ricezione dati di Input;
.. stampa ciclica del numero di copie richieste.

Avendo deciso di ingrandire i caratteri dell'intestazione del foglio, abbiamo dovuto controllare che il numero dei caratteri non superi 40. In caso si hanno 3 possibili STOP:
.. linea 194 se Titolo+Nome+Cognome (M$(1), M$(2), M$(3));
.. linea 344 per l'indirizzo (M$(4), M$(5), M$(6));
.. linea 355 per il telefono (M$(7));
e si può intervenire in immediato per accorciare le relative stringhe e poi continuare con CONT. Osserva come otteniamo la centratura con l'aggiunta del giusto numero di spazi a sinistra.

Segue uno schema o blocchi non molto dettagliato del programma.

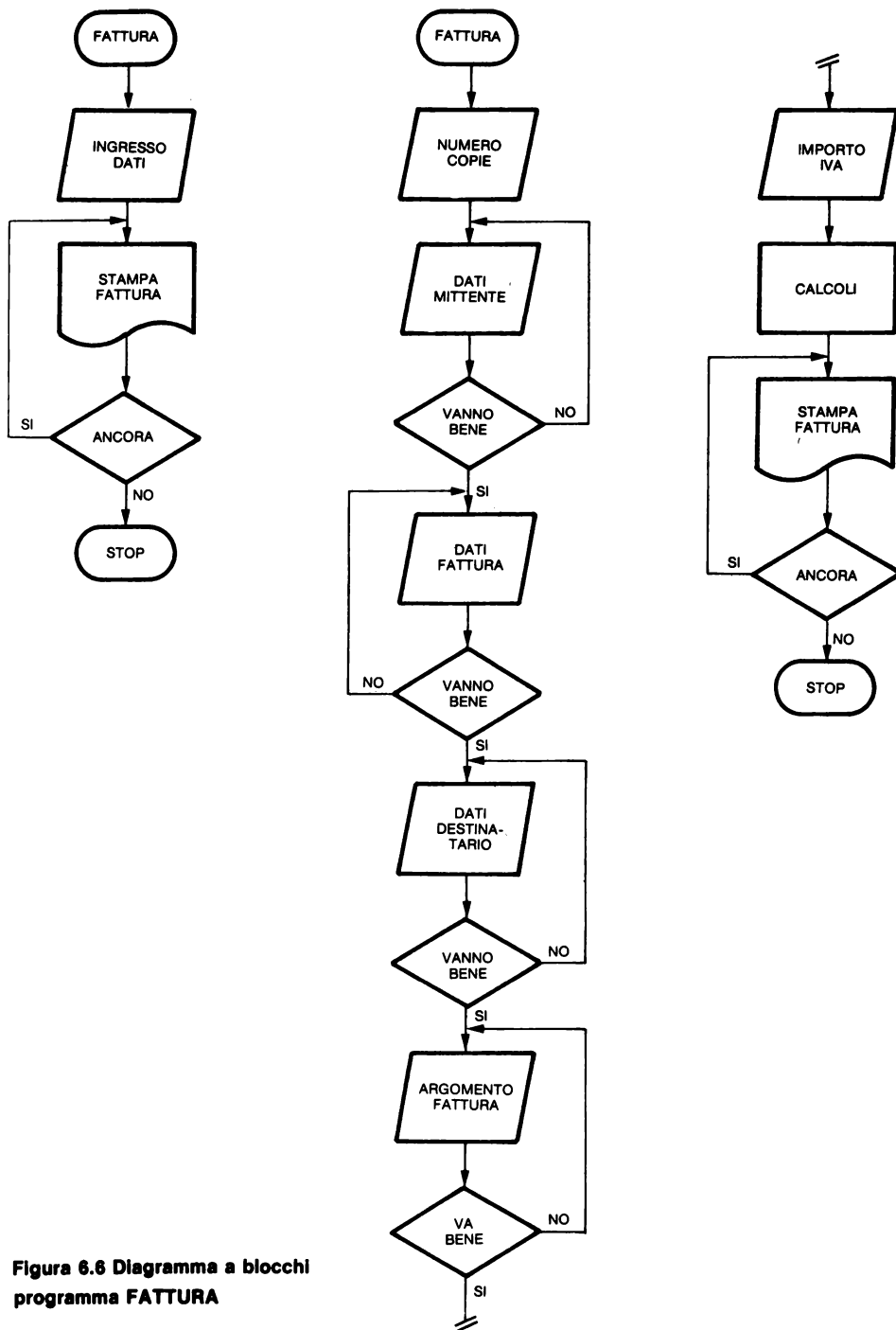



Figura 6.6 Diagramma a blocchi programma FATTURA

Segue il listato del programma FATTURA.

```
1 REM FATTURA
5 DIMM$(10),I$(4),C$(10),CI$(4)
6 C$(1)="TITOLO: ":C$(2)="NOME: "
7 C$(3)="COGNOME: "
8 C$(4)="CAP: ":C$(5)="CITTA': "
9 C$(6)="INDIRIZZO: ":C$(7)="TEL.: "
10 C$(8)="COD. FISCALE: "
11 C$(10)="LUOGO/DATA NASCITA: "
12 S3$="----------------------------":L$="L. "
13 C$(9)="PARTITA IVA: "
14 CI$(3)=C$(6):CI$(4)="CAP/CITTA': "
15 S$="":FORK=1TO40:S$=S$+" ":NEXTK
16 INPUT"♥QQQUANTE COPIE: ";NF
17 IFNF<0THEN16
19 V$="SQQQQQQQQQQQQQQQQQQQQQQQQ"
20 S1$="♥QQDATI MITTENTEQQ"
23 CI$(1)=C$(1):CI$(2)="RAG. SOCIALE: "
25 PRINTS1$:FORK=1TO10
30 GOSUB600
31 PRINTK$"-"C$(K);:INPUTM$(K)
33 NEXTK
35 GOSUB1000
40 IFR$="S"THEN90
45 INPUT"QUALE CAMPO: ";N
50 IFN<1ORN>10THENPRINT"]]":GOTO45
55 PRINTS1$:IFN=1THEN63
56 FORK=1TON-1
57 GOSUB600
60 PRINTK$" "C$(K)M$(K):NEXTK
63 GOSUB600
65 PRINTK$" "C$(N);:INPUTM$(N)
70 IFN=10THEN35
75 FORK=N+1TO10
76 GOSUB600
77 PRINTK$"-"C$(K)M$(K):NEXTK
80 GOTO35
90 PRINT"♥QQDATI FATTURAQQ"
93 INPUT"CITTA' FATTURA: ";CF$
95 INPUT"DATA: ";DF$
97 INPUT"NUM. FATT.: ";N$
```

```
99 GOSUB1000:IFR$="S"THEN103
100 GOSUB90
103 S2$="▒▒▒DATI DESTINATARIO▒▒"
105 PRINTS2$
107 FORK=1TO4
109 PRINTK;CI$(K);:INPUTI$(K)
111 NEXTK
113 GOSUB1000
115 IFR$="S"THEN133
117 INPUT"QUALE CAMPO:  ";N
119 IFN<1ORN>4THENPRINT"▒▒":GOTO117
121 PRINTS2$:IFN=1THEN125
122 FORK=1TON-1
123 PRINTK;CI$(K)I$(K):NEXTK
125 PRINTN;CI$(N);:INPUTI$(N)
127 IFN=4THEN113
129 FORK=N+1TO4:PRINTK;CI$(K)I$(K):NEXTK
131 GOTO113
133 PRINT"▒▒▒ARGOMENTO FATTURA▒▒"
135 INPUT"▒QUANTE LINEE:  ";ZF
137 IFZF>10THENPTRINT"▒▒":GOTO135
139 PRINT"▒";:FORK=1TOZF
141 PRINTK;":  ";:INPUTZ$(K)
143 IFLEN(Z$(K))>50THENZ$(K)=LEFT$(Z$(K),50)
147 NEXTK
149 PRINT"▒";:FORK=1TOZF
151 PRINTK;"-";Z$(K):NEXTK
153 GOSUB1000
155 IFR$="S"THEN165
157 INPUT"QUALE CAMPO:  ";N
159 IFN<1ORN>ZFTHENPRINT"▒▒":GOTO157
161 PRINT"RISCRIVI RIGA":INPUTZ$(N)
162 IFLEN(Z$(N))>50THENZ$(N)=LEFT$(Z$(N),50)
163 GOTO149
165 INPUT"IMPORTO:  ";M:ZX=M
167 GOSUB500:L=LEN(A$):M$=A$
169 INPUT"▒▒% IVA:  ";IV
171 ZX=INT(M*IV/100):GOSUB500
173 IV$=RIGHT$(S$+A$,L)
175 X=M+ZX:ZX=X:GOSUB500
177 TF$=RIGHT$(S$+A$,L)
179 PRINT"▒▒▒POSIZIONA IL FOGLIO DI CARTA"
```

```
181 PRINT"PREMI UN TASTO PER CONTINUARE"
183 GETR$:IFR$=""THEN183
190 OPEN4,4
191 FORKF=1TONF
192 A$=M$(1)+" "+M$(2)+" "+M$(3)
194 X=LEN(A$):IFX>40THENSTOP
198 X=(80-X*2)/4
201 IFX=0THEN206
203 A$=LEFT$(S$,X)+A$
206 PRINT#4,CHR$(14)A$CHR$(15)
208 I=7:GOSUB1100
210 FORK=1TO4:PRINT#4,I$(K):NEXTK
220 I=2:GOSUB1100
225 PRINT#4,TAB(50)"NUMERO FATTURA:"N$
230 PRINT#4,TAB(50)CF$", "DF$
235 I=6:GOSUB1100:C=22
255 FORK=1TOZF
260 PRINT#4,Z$(K)
265 NEXTK:C=C+ZF
300 PRINT#4,TAB(50)L$M$:C=C+1
315 PRINT#4
320 PRINT#4," I V A  ";IV;"%";TAB(37)L$IV$
325 PRINT#4,TAB(40);S3$:C=C+2
329 PRINT#4,TAB(40)"TOTALE:    "L$TF$
330 I=13:GOSUB1100:C=C+14
335 PRINT#4,"DATI ANAGRAFICI:"
336 PRINT#4,M$(2)" "M$(3)
337 PRINT#4,"NASCITA: "M$(10)
338 PRINT#4,"RESIDENZA: "M$(4)"-"M$(5)"-"M$(6)
339 PRINT#4,C$(8)M$(8)
340 PRINT#4,C$(9)M$(9):C=C+5
341 I=58-C:GOSUB1100
343 A$=M$(4)+"-"+M$(5)+"-"+M$(6)
344 X=LEN(A$):IFX>40THENSTOP
345 X=(80-X*2)/4:IFX=0THEN348
346 A$=LEFT$(S$,X)+A$
348 PRINT#4,CHR$(14)A$CHR$(15)
350 A$="TEL.: "+M$(7):X=LEN(A$)
355 IFX>40THENSTOP
360 X=(80-X*2)/4:IFX=0THEN370
365 A$=LEFT$(S$,X)+A$
370 PRINT#4,CHR$(14)A$CHR$(15)
```

```
375 I=4:GOSUB1100
380 NEXTKF:CLOSE4:STOP
500 MA$=STR$(ZX):X=LEN(MA$)-1
501 IFX<=3THENA$=MA$:GOTO535
505 Z=INT(X/3):Y=X-Z*3:A$=""
510 IFY=0THEN520
515 A$=A$+LEFT$(MA$,Y+1)+"."
520 A$=A$+MID$(MA$,Y+2,3)
525 Z=Z-1:IFZ=0THEN535
530 A$=A$+".":Y=Y+3:GOTO520
535 RETURN
600 K$=STR$(K):IFLEN(K$)=2THENK$=" "+K$
601 RETURN
1000 PRINTV$
1005 INPUT"CONFERMI (S/N):";R$:RETURN
1100 FORK=1TOI:PRINT#4:NEXTK:RETURN
```

Segue una fattura risultato del programma fotografata rimpicciolendola.

SIG. GIULIO BELLINI

SPETT.
DITTA BELLA
VIA CAMPAGNA, 57
110111 COMO

NUMERO FATTURA:39
VOGHERA, 15 GENNAIO 1984

PER LAVORI DI RICERCA DA VOI COMMISSIONATI
COME DA ACCORDI INTERCORSI

L. 600.000

I V A 18 % L. 108.000

TOTALE: L. 708.000

DATI ANAGRAFICI:
GIULIO BELLINI
NASCITA: VERONA-1/1/1947
RESIDENZA: 110111-VOGHERA-VIA SPERANZA, 56
COD. FISCALE: 01000010111
PARTITA IVA: 100010100011

110111-VOGHERA-VIA SPERANZA, 56
TEL.: 1011101011

Nello svolgimento del programma troverai applicati gli argomenti oggetto del Capitolo 3 e del Capitolo 5; in particolare osserva come viene gestito il contatore delle righe di stampa C per poter andare correttamente a nuovo foglio. Inoltre è interessante il sottoprogramma che inizia in 500; esso serve per mettere i puntini separatori nel numero da stampare. Nel rispondere alle richieste di dati, se vuoi che il programma accetti virgole e due punti (per esempio per l'indirizzo, dove si usa far precedere da una virgola il numero civico) devi ricordarti di scrivere il carattere "virgolette" prima del primo carattere valido e subito dopo l'ultimo, cioè scrivere la risposta tra virgolette.
Se vuoi usare il programma per mandare le tue fatture ai clienti principali puoi modificarlo in questo modo:
.. trasformare le linee da 23 a 80 in 10 linee di assegnazione delle 10 stringhe M$, cancellando quelle in più; così i tuoi dati diventano costanti di programma;
.. trasformare le linee da 105 a 131 in 4 linee di assegnazione delle 4 stringhe I$, cancellando quelle in più; così i dati del destinatario diventano costanti di programma;
.. memorizzare una copia del programma su disco o su nastro, assegnandogli un nome di riconoscimento;
.. ripetere le ultime due operazioni per il numero di copie diverse di programma che desideri avere.
In tale modo, per inviare una fattura a un determinato cliente, basta richiamare il relativo programma e introdurre solo i dati della fattura e il suo argomento.

CAPITOLO 7

CODICI E NUMERI DEL CALCOLATORE

7.1 PREMESSA

In un byte si possono rappresentare numeri decimali da 0 a 255.
L'interpretazione che viene data a un byte dipende dal contesto nel quale esso viene preso in esame.
Nell'interpretazione in codice ASCII, una parte di questi 256 numeri corrisponde a caratteri stampabili, una parte degli altri corrisponde a specifiche azioni (codici di controllo), alcuni non hanno effetto.
Il contenuto dei byte può rappresentare istruzioni di programma in linguaggio macchina; in questo caso l'interpretazione viene fatta in un modo particolare e non intendiamo occuparcene in questo libro.
Un byte può contenere un codice che corrisponde a una istruzione del linguaggio Basic.
Uno o più byte associati possono contenere dei numeri espressi in binario.

7.2 NUMERI INTERI

Nell'interpretazione numerico binaria, due byte associati (byte basso + byte alto) contengono un numero intero che, se considerato senza segno, può raggiungere il valore 65535; infatti 255 + 256x255 = 65535. Questo modo è normalmente usato per rappresentare indirizzi di memoria.

Se il numero definito da due byte associati è considerato come numero con segno, allora il primo bit di sinistra del byte alto serve per il segno. Il numero positivo ha questo bit a zero, seguito dai bit che danno il valore assoluto del numero. Il numero negativo ha invece questo bit a uno, seguito dai bit che danno il valore del numero rappresentato in complemento a 2.

Il numero positivo può variare da 0 a 32767; infatti 255 + 256x127 = 32767 (il byte

alto può raggiungere solo il valore 127, dato che il bit di sinistra, bit 7, è usato per il segno). Per esempio:

numero decimale: 35
contenuto dei 2 byte binari: 000000000100011

numero decimale: 32767
contenuto dei 2 byte binari: 011111111111111

numero decimale: 0
contenuto dei 2 byte binari: 0000000000000000

Il numero negativo può variare da -32768 a -1. Per arrivare alla rappresentazione binaria di un numero intero negativo si può procedere così:
.. si considera la potenza del 2 immediatamente superiore al massimo numero rappresentabile (con 15 bit a disposizione, 8 del byte basso e 7 del byte alto, si arriva al massimo al valore posizionale di 2 elevato a 14); in questo caso 2 elevato a 15, che è uguale a 32768;
.. si considera il valore assoluto del numero che si vuole rappresentare;
.. si calcola 32768 - (valore assoluto numero);
.. si scrive in binario il risultato del calcolo precedente e si aggiunge un bit 1 a sinistra per il segno.

Esempio: numero decimale da rappresentare = -35
calcolo: 32768 - 35 = 32733
rappresentazione binaria di 32733 ottenuta per sottrazioni successive delle potenze di 2:

32733 -	16384 =	16349	dà un bit 1 in posizione	14
16349 -	8192 =	8157	dà un bit 1 in posizione	13
8157 -	4096 =	4061	dà un bit 1 in posizione	12
4061 -	2048 =	2013	dà un bit 1 in posizione	11
2013 -	1024 =	989	dà un bit 1 in posizione	10
989 -	512 =	477	dà un bit 1 in posizione	9
477 -	256 =	221	dà un bit 1 in posizione	8
221 -	128 =	93	dà un bit 1 in posizione	7
93 -	64 =	29	dà un bit 1 in posizione	6
29 -	16 =	13	dà un bit 1 in posizione	4
13 -	8 =	5	dà un bit 1 in posizione	3
5 -	4 =	1	dà un bit 1 in posizione	2
1 -	1 =	0	dà un bit 1 in posizione	0

il contenuto dei 2 byte risulta quindi, aggiungendo il bit 1 di segno negativo in

posizione15 (si considera posizione 15 quella più a sinistra e posizione 0 quella più a destra):

    1111111111011101

Si può arrivare rapidamente allo stesso risultato applicando questa semplice regola:

.. scrivere in binario il valore assoluto del numero, cioè: 35, che equivale a

    0000000000100011

.. sostituire ai bit 0 dei bit 1 e ai bit 1 dei bit 0, alla fine sommare un bit 1 nella posizione 0; infatti:

con la sostituzione si ha	1111111111011100
aggiungendo 1 in fondo si ha	1111111111011101.

Se consideriamo il numero -1, otteniamo in binario: 1111111111111111, mentre se consideriamo il numero -32768, otteniamo: 1000000000000000.

Con il programma INTINMEM che segue si stampa sul video e sulla stampante un numero intero con segno e il contenuto dei 7 byte che lo rappresentano in memoria (vedi Capitolo 8).

```
1 REM NUMERI INTERI IN MEMORIA
2 INPUT"{CLR}{DOWN}{DOWN}NUMERO INTERO = ";N%
5 M$="NUMERI INTERI IN MEMORIA"
10 A=256*PEEK(46)+PEEK(45)
15 INPUT"{CLR}{DOWN}{DOWN}{DOWN}VUOI USARE LA STAMPANTE (S/N) ";R$
17 PRINT"{CLR}{DOWN}"M$:PRINT"{DOWN}N% = ";N%
40 FORK=0TO6:PRINTPEEK(A+K);" ";:NEXTK:PRINT
50 IFR$="N"THEN100
55 OPEN4,4:CMD4:PRINTM$
60 FORK=0TO6:PRINTPEEK(A+K);" ";:NEXTK
70 PRINT:PRINT"N%=";N%:PRINT#4:CLOSE4
100 STOP
```

In questo programma definiamo come prima variabile il numero N% (intero), in modo che si trovi in memoria a partire dalla prima posizione della zona delle variabili (linea 10, calcolo del puntatore A all'inizio della zona variabili). Viene chiesto un numero intero alla linea 2 e poi viene evidenziato il contenuto dei 7 byte che lo contengono in memoria. Il contenuto dei 7 byte è il seguente:

..primo: il numero 206 che è la somma del codice ASCII di N, cioè il numero 78, con 128 (206=78+128);
..secondo: il numero 128, dato che il nome della variabile è formato da una sola lettera;
..terzo e quarto: il valore del numero in binario;
..quinto, sesto e settimo: zero, dato che il numero rappresentato è intero.

Dagli esempi che seguono puoi verificare quanto abbiamo esposto sopra.

RISULTATI PROGRAMMA PRECEDENTE

```
NUMERI INTERI IN MEMORIA
 206   128   127   255   0   0   0
N%= 32767

NUMERI INTERI IN MEMORIA
 206   128   128   1   0   0   0
N%=-32767

NUMERI INTERI IN MEMORIA
 206   128   0   0   0   0   0
N%= 0

NUMERI INTERI IN MEMORIA
 206   128   255   255   0   0   0
N%=-1
```

7.3 NUMERI REALI

Per rappresentare i numeri reali (non interi, numeri con cifre dopo il punto decimale), che nel calcolatore vengono memorizzati in forma esponenziale, si usano 5 bytes consecutivi. Di questi 5 bytes il primo contiene l'esponente (caratteristica) e gli altri quattro la mantissa. Ricordiamo che un numero in forma esponenziale (floating point) è formato dalle cifre significative del numero (mantissa) e dall'esponente (caratteristica) da usare per calcolare il valore del numero. Nell'aritmetica decimale l'esponente va applicato alla base 10, mentre nell'aritmetica

binaria esso va applicato alla base 2. Il calcolo per ottenere il valore del numero in decimale è il seguente:

numero = mantissa x 10 elevato a esponente.

In questo calcolatore la mantissa viene conservata nella forma: .xxxxxxxxxx (forma normalizzata), considerando solo le cifre significative, e l'esponente ha il valore necessario per calcolare il numero. I limiti in decimale sono: per l'esponente da -39 a +38, e per la mantissa da -42949673 a +42949673. In binario questi limiti diventano per l'esponente da -128 a +127. Per poter gestire un esponente, sia positivo che negativo, usando un solo byte, si usa la convenzione che lo zero per l'esponente è spostato al valore 128; per questa ragione gli esponenti positivi vanno da 128 a 255 e quelli negativi da 0 a 127.
Per calcolare i limiti per la mantissa, si deve considerare il valore ottenibile associando 4 byte consecutivi; si hanno a disposizione 31 bit per il numero e il primo bit a sinistra del byte più alto per il segno. Nel nostro calcolatore, dato che la mantissa del numero viene normalizzata in modo che la prima cifra a sinistra (a destra del punto decimale) è sempre un bit 1, tale bit viene eliminato dalla memoria, ma se ne tiene conto nei calcoli. In conseguenza la mantissa può essere formata da 4 byte contenenti in tutto 32 bit 1. Prima di memorizzare il numero, il primo bit a sinistra viene eliminato e sostituito dal bit del segno, 0 per i numeri positivi e 1 per i numeri negativi. In questo caso, contrariamente a quanto avviene per i numeri interi, i numeri negativi sono memorizzati in valore assoluto, con aggiunto all'inizio il bit 1 di segno.
Si rappresentano in questo modo anche i numeri interi che cadono fuori dell'intervallo —32768/+ 32767.
Nel programma FLOATINMEM che segue, con una tecnica analoga a quella del programma precedente, cioè definendo per prima cosa la variabile N, si ottiene che essa compaia per prima nella zona variabili. Servendosi del puntatore P (calcolato alla linea 4) si leggono dalla memoria con la funzione PEEK i 5 byte che rappresentano il numero; il primo la caratteristica (esponente) e gli altri quattro la mantissa. Il programma stampa il numero introdotto, il contenuto in decimale e in binario dei 4 byte della mantissa, la stringa binaria che costituisce la mantissa come è stata letta inizialmente e quella con aggiunto il primo bit 1 al posto del bit di segno. Viene poi stampato l'esponente come viene letto e come risulta calcolato sottraendo 128, il valore binario della parte intera del numero e il valore binario della parte decimale, ambedue senza segno, e, infine il numero ricalcolato. Puoi fare diverse prove e vedrai che, se superi i limiti consentiti per i numeri, compare una segnalazione di OVERFLOW. In certi casi si hanno differenze tra il numero introdotto e il numero ricalcolato; esse sono dovute ad errori di arrotondamento.
Inoltre i numeri vengono stampati con 9 cifre, mentre in memoria ne sono conservate 10.

```
1 REM CALCOLO NUMERI FLOATING POINT
2 PRINT"♥QQSCRIVI UN NUMERO DECIMALE":INPUTN
3 IFSW=1THEN8
4 P=PEEK(45)+256*PEEK(46):GOSUB300
5 INPUT"VUOI USARE LA STAMPANTE (S/N) ";R$
6 IFR$="S"THENOPEN4,4:PRINT#4,P$:PRINT#4
8 PRINT"♥QQQ"C1$;N:L$=""
9 IFR$="S"THENPRINT#4:PRINT#4,C1$;N
11 IFPEEK(P)<>78THENSTOP
12 FORI=1TO5:X(I)=PEEK(P+I+1):NEXTI
13 PRINTC2$
14 FORI=2TO5:PRINTX(I);"        ";:NEXTI
15 IFR$="N"THEN19
16 PRINT#4,C2$
17 FORI=2TO5:PRINT#4,X(I);"        ";
18 NEXTI:PRINT#4
19 PRINT:PRINTO$
20 IFR$="S"THENPRINT#4,O$
30 FORI=2TO5:GOSUB100:PRINTF$(I);" ";:NEXTI
40 IFR$="N"THEN 60
45 FORK=2TO5:PRINT#4,F$(K);" ";:NEXTK:PRINT#4
60 PRINT
62 PRINTC3$:PRINTL$
63 IFR$="S"THENPRINT#4,C3$:PRINT#4,L$
68 S=1+2*(LEFT$(F$(2),1)="1")
70 F$(2)="1"+RIGHT$(F$(2),7)
72 PRINTN1$:PRINTN2$
73 IFR$="S"THENPRINT#4,N1$:PRINT#4,N2$
75 LL$=F$(2)+F$(3)+F$(4)+F$(5)
76 PRINTLL$
77 IFR$="S"THENPRINT#4,LL$
79 Y=X(1)-128
80 PRINTC8$;X(1)
81 IFR$="N"THEN83
82 PRINT#4,C8$;X(1)
83 PRINTC5$;Y
84 IFR$="S"THENPRINT#4,C5$;Y
85 IFY<0THENM$="0":D$=LEFT$(Z$,-Y)+LL$:GOTO92
86 IFX(1)=0THENM$="0":D$="0":GOTO92
87 IFY>LEN(LL$)THENLL$=LEFT$(LL$+Z$,Y)
89 M$=LEFT$(LL$,Y):Z=LEN(LL$)-Y
90 IFZ<=0THEND$="0":GOTO92
```

```
91 D$=RIGHT$(LL$,(LEN(LL$)-Y))
92 PRINTC6$,M$:PRINTC7$,D$:IFR$="N"THEN95
93 PRINT#4,C6$:PRINT#4,M$
94 PRINT#4,C7$:PRINT#4,D$
95 GOSUB200:GOSUB250:PRINTC4$;(M+D)*S
96 IFR$="S"THENPRINT#4,C4$;(M+D)*S
97 INPUT"■■ANCORA (S/N) ";B$
98 IFB$="S"THENSW=1:GOTO2
99 CLOSE4:STOP
100 F$(I)="":FORJ=7TO0STEP-1
101 IFINT(X(I)/2↑J)=0THEN105
103 F$(I)=F$(I)+"1":X(I)=X(I)-2↑J:GOTO110
105 F$(I)=F$(I)+"0"
110 NEXT:L$=L$+F$(I):RETURN
156 PRINTD;" ";
200 M=0:IFM$="0"THENRETURN
201 FORJ=LEN(M$)TO1STEP-1
203 IFMID$(M$,J,1)="1"THENM=M+2↑(LEN(M$)-J)
205 NEXTJ:RETURN
250 D=0:IFD$="0"THENRETURN
253 FORJ=1TOLEN(D$)
255 IFMID$(D$,J,1)="1"THEND=D+(1/2)↑J
260 NEXTJ:RETURN
300 N1$="STRINGA BIT CON AGGIUNTO BIT INIZIALE"
301 N2$="AL POSTO DEL BIT DI SEGNO"
302 O$="QUATTRO BYTE MANTISSA IN BINARIO"
303 P$="NUMERI FLOATING POINT"
304 C1$="NUMERO INTRODOTTO = "
306 C2$="QUATTRO BYTE MANTISSA IN MEMORIA"
307 C3$="STRINGA DI BIT INIZIALE"
308 C4$="NUMERO CALCOLATO = "
309 C5$="ESP. PER BASE 2 = "
310 C6$="VAL.BIN.PART.INT. SENZA SEGNO"
311 C7$="VAL.BIN.PART.DEC. SENZA SEGNO"
312 C8$="BYTE ESP. = "
313 Z$="0":FORK=1TO127:Z$=Z$+"0":NEXTK
350 RETURN
```

RISULTATI PROGRAMMA PRECEDENTE

```
NUMERI FLOATING POINT


NUMERO INTRODOTTO =  2.93873588E-39
QUATTRO BYTE MANTISSA IN MEMORIA
 0         0         0         0
QUATTRO BYTE MANTISSA IN BINARIO
00000000 00000000 00000000 00000000
STRINGA DI BIT INIZIALE
00000000000000000000000000000000
STRINGA BIT CON AGGIUNTO BIT INIZIALE
AL POSTO DEL BIT DI SEGNO
10000000000000000000000000000000
BYTE ESP. =  1
ESP. PER BASE 2 = -127
VAL.BIN.PART.INT. SENZA SEGNO
0
VAL.BIN.PART.DEC. SENZA SEGNO
000000000000000000000000000000000000000000000000
                100000000000000000000000000000000
000000000000000000000000000000000000000000000001
                 0000000000000000000000000000000

NUMERO CALCOLATO =  0


NUMERO INTRODOTTO =  4.2949673E+09
QUATTRO BYTE MANTISSA IN MEMORIA
 0         0         0         2
QUATTRO BYTE MANTISSA IN BINARIO
00000000 00000000 00000000 00000010
STRINGA DI BIT INIZIALE
00000000000000000000000000000010
STRINGA BIT CON AGGIUNTO BIT INIZIALE
AL POSTO DEL BIT DI SEGNO
10000000000000000000000000000010
BYTE ESP. =  161
ESP. PER BASE 2 =  33
VAL.BIN.PART.INT. SENZA SEGNO
100000000000000000000000000000100
```

```
VAL.BIN.PART.DEC. SENZA SEGNO
0
NUMERO CALCOLATO =  4.2949673E+09

NUMERO INTRODOTTO =  1E-36
QUATTRO BYTE MANTISSA IN MEMORIA
 42          36          36          154
QUATTRO BYTE MANTISSA IN BINARIO
00101010 00100100 00100100 10011010
STRINGA DI BIT INIZIALE
00101010001001000010010010011010
STRINGA BIT CON AGGIUNTO BIT INIZIALE
AL POSTO DEL BIT DI SEGNO
10101010001001000010010010011010
BYTE ESP. =  9
ESP. PER BASE 2 = -119
VAL.BIN.PART.INT. SENZA SEGNO
0
VAL.BIN.PART.DEC. SENZA SEGNO
000000000000000000000000000000000000000000000000
                                00000000000000000000000000000000
000000000000000000000000000000000000000101010100
                         01001000010010010011010
NUMERO CALCOLATO =  9.991702E-37
```

Come puoi vedere dai risultati riportati, il numero 2.93873588E-39 dà come risultato nel ricalcolo il valore 0, mentre il numero 4.2949673E+09 viene accettato. Ti ricordo che la mantissa fornisce le cifre significative del numero, mentre la caratteristica influisce sull'ordine di grandezza del numero.

7.4 CODICI ASCII E D/CODE - CARATTERI STAMPABILI

Ora ci occupiamo di quello che viene stampato sul video.

Al momento dell'accensione del calcolatore, oppure dopo aver premuto contemporaneamente i tasti RUN/STOP e RESTORE, oppure dopo aver scritto: SYS 64738 seguito da RETURN, le condizioni di colore del video sono:
.. bordo AZZURRO, corrispondente al codice colore 14;
.. sfondo BLU, corrispondente al codice colore 6;
.. cursore AZZURRO, corrispondente al codice colore 14.

Se, in queste condizioni del calcolatore, si vanno a leggere i contenuti dei byte che controllano lo stato dei colori del video si trova:
.. 254 nel byte 53280 che controlla il COLORE DEL BORDO, cioè il codice 14 del colore azzurro più 240;
.. 246 nel byte 53281, che controlla il COLORE DELLO SFONDO, cioè il codice 6 del colore blu più 240;
.. 14 nel byte 646, che controlla il COLORE DEL CURSORE e quindi del CARATTERE, cioè il codice del colore azzurro.

Il calcolatore dispone di due SET di caratteri in codice ASCII; questo codice è usato per la rappresentazione interna dei caratteri. I caratteri sono invece codificati in D/CODE per poter apparire sul video; il D/CODE fa da puntatore alla descrizione del cara.tere per punti.
Ogni SET di caratteri ne comprende 256 (256 codici diversi); di questi 256 caratteri solo 128 sono stampabili, ma in due modi: diretto e inverso, scambiando cioè il colore dello sfondo con quello del carattere (positivo e negativo). Il D/CODE ha quindi 256 codici, 128 per i caratteri diretti e 128 per i caratteri inversi, in ogni set di caratteri.
Sulla tastiera si vedono solo 64 caratteri stampabili, ma essi diventano il doppio usando ogni tasto anche insieme al tasto SHIFT.

La relazione tra codice ASCII e D/CODE è la seguente:

ASCII	D/CODE
32 <= A <= 63	D = A
64 <= A <= 95	D = A - 64
96 <= A <= 127	D = A - 32
160 <= A <= 191	D = A - 64
192 <= A <= 254	D = A - 128
A = 255	D = A - 161

Nelle condizioni di inizializzazione il byte 53272 contiene il numero 21; tale numero serve per selezionare il SET di caratteri MAIUSCOLO/GRAFICO. Se si scrive in questo byte il numero 23 (POKE 53272,23), si ottiene di passare al SET di caratteri MAIUSCOLO/MINUSCOLO. Lo stesso effetto si ottiene premendo contemporaneamente i due tasti COMMODORE e SHIFT. Si può passare da un SET all'altro di caratteri scrivendo:

PRINT CHR$(14) per ottenere il set maiuscolo/minuscolo
PRINT CHR$(142) per ottenere il set maiuscolo/grafico.

In realtà nel byte 53272 il numero 21 e 23 selezionano il set di caratteri, ma anche la posizione di inizio della mappa video a 1024.

I due SET di caratteri (MAPPA CARATTERI) si trovano in ROM dal byte 53248 al byte 57343 e occupano 4096 byte. Infatti per descrivere ogni carattere occorrono 8 byte (8 x 8 punti, 1 bit per punto); 256x8=2048 byte per ogni SET. Quando i bit di posizione 2 e 3 del byte 53272 sono a "10" viene selezionato il SET che inizia a 53248, mentre quando sono a "11" viene selezionato il SET che inizia a 55296.

Con il programma che segue si va a prelevare dalla MAPPA CARATTERI la descrizione di un carattere per punti (bit 0 e 1) e lo si disegna sul video e sulla stampante ponendo al posto di un punto (quello che avviene di norma in modo automatico; 1 bit 1 dà luogo a un punto) un asterisco; inoltre si scrive il valore di ogni byte, che è servito per disegnare il carattere ingrandito, in decimale e in binario.

```
0 REM DISEGNO CARATTERI
1 INPUT"⌉■RISULTATI SU STAMPANTE? (S/N) ";R$
2 PRINT:IFR$="N"THEN6
3 OPEN4,4:PRINT#4
4 PRINT#4,"***STAMPA IMMAGINE CARATTERI***"
5 PRINT#4
6 D$="":PRINT"QUALE SET VUOI USARE: "
7 PRINT"  1  PER SET MAIUSCOLO/GRAFICO"
8 PRINT"  2  PER SET MAIUSCOLO/MINUSCOLO"
9 INPUT"■■  OPZIONE:";D$:IFD$=""THEN100
10 IFD$<>"1"ANDD$<>"2"THENPRINT"TTT]":GOTO9
14 M$=" SET MAIUSCOLO/MINUSCOLO"
15 IFD$="1"THENM$=" SET MAIUSCOLO/GRAFICO"
16 PRINT"■■SCRIVI IL CODICE DEL CARATTERE"
17 PRINT"IN D/CODE";:INPUTD:PRINT
18 IFD<0ORD>255THENPRINT"TTTTT]":GOTO14
19 POKE56334,PEEK(56334)AND254
20 POKE1,PEEK(1)AND251
21 A=53248+(VAL(D$)-1)*2048
22 FORK=0TO7:D(K)=PEEK(A+D*8+K):NEXTK
23 POKE1,PEEK(1)OR4:POKE56334,PEEK(56334)OR1
24 PRINT"⌉■■■D/CODE=";D;M$:PRINT
25 PRINT"CARATTERE CORRISP. AGLI 8 BYTES:"
26 PRINT
27 PRINT"CARATTERE  VAL. BINARIO  VAL. DEC."
28 PRINT:PRINT:IFR$="N"THEN34
29 PRINT#4,"D/CODE=";D;M$:PRINT#4
30 PRINT#4,"CARATTERE CORRISP. AGLI 8 BYTES:"
31 PRINT#4
```

```
32 PRINT#4,"  CARATTERE    VAL. BINARIO   ";
33 PRINT#4,"VAL. DEC.":PRINT#4
34 FORK=0TO7:D$(K)="":E$(K)="":B=D(K)
35 FORJ=0TO7:IFINT(B/2↑(7-J))=0THEN40
36 D$(K)=D$(K)+"*":E$(K)=E$(K)+"1"
39 B=B-2↑(7-J):GOTO41
40 D$(K)=D$(K)+" ":E$(K)=E$(K)+"0"
41 NEXTJ:NEXTK:FORK=0TO7
42 PRINTD$(K);"     ";E$(K);"      ";D(K)
43 IFR$="N"THEN50
44 PRINT#4," ";D$(K);"        ";
45 PRINT#4,E$(K);"      ";D(K)
50 NEXTK
100 IFR$="S"THENCLOSE4
110 STOP
```

RISULTATI DEL PROGRAMMA PRECEDENTE

```
***STAMPA IMMAGINE CARATTERI***

D/CODE= 65  SET MAIUSCOLO/GRAFICO

CARATTERE CORRISP. AGLI 8 BYTES:

  CARATTERE     VAL. BINARIO   VAL. DEC.

       *         00001000       8
      ***        00011100       28
     *****       00111110       62
    *******      01111111       127
    *******      01111111       127
      ***        00011100       28
     *****       00111110       62
                 00000000       0


***STAMPA IMMAGINE CARATTERI***

D/CODE= 193  SET MAIUSCOLO/GRAFICO

CARATTERE CORRISP. AGLI 8 BYTES:

  CARATTERE     VAL. BINARIO   VAL. DEC.

   **** ***      11110111       247
   ***   **      11100011       227
   **     *      11000001       193
   *             10000000       128
   *             10000000       128
   ***   **      11100011       227
   **     *      11000001       193
   ********      11111111       255
```

```
***STAMPA IMMAGINE CARATTERI***

D/CODE= 88  SET MAIUSCOLO/MINUSCOLO

CARATTERE CORRISP. AGLI 8 BYTES:

  CARATTERE      VAL. BINARIO    VAL. DEC.

   **  **          01100110        102
   **  **          01100110        102
    ****           00111100        60
     **            00011000        24
    ****           00111100        60
   **  **          01100110        102
   **  **          01100110        102
                   00000000        0

***STAMPÁ IMMAGINE CARATTERI***

D/CODE= 193  SET MAIUSCOLO/MINUSCOLO

CARATTERE CORRISP. AGLI 8 BYTES:

  CARATTERE      VAL. BINARIO    VAL. DEC.

  ***  ***         11100111        231
  **    **         11000011        195
  *  **  *         10011001        153
  *      *         10000001        129
  *  **  *         10011001        153
  *  **  *         10011001        153
  *  **  *         10011001        153
  ********         11111111        255
```

Se osservi i risultati precedenti puoi notare che scegliendo lo stesso set di caratteri e usando codici D/CODE che differiscono di 128 si ottiene lo stesso carattere in positivo e negativo.
Il disegno dei caratteri ingranditi corridponde esattamente al disegno dei caratteri per punti sul video. I caratteri stampati dalla stampante differiscono come forma da quelli del video; infatti essi dipendono dalla mappa dei caratteri memorizzata nella ROM della stampante stessa.
Nel programma precedente, la MAPPA CARATTERI è stata prelevata dalla ROM

a partire dall'indirizzo 53248. Per poter operare così si è dovuto sospendere l'uso della tastiera e disattivare le interruzioni automatiche con il primo comando POKE alla linea 19, poi con il secondo comando POKE si è sospeso l'Input/Output normale e resa disponibile la mappa in ROM. A questo punto si sono prelevati i caratteri dalla ROM e poi si sono ripristinate le condizioni normali alla linea 23. Rimandiamo al Capitolo 8 per rivedere l'argomento.
Gli 8 byte che descrivono un carattere sono memorizzati nel vettore D(k). I vettori D(k), D$(k) e E$(k) non sono dimensionati con una DIM, dato che bastano 8 elementi. D$(k) serve per disegnare il carattere con gli asterischi, mentre E$(k) serve per costruire come stringa il valore binario del byte. Nelle righe da 34 a 41 viene fatta la decodifica del valore di ogni byte per cercare i bit 1 presenti; si ottiene questo confrontando la funzione INT del valore del byte diviso per le successive potenze decrescenti di 2, partendo da 2 elevato a 7, con 1 e, nel caso si sia trovato 1, decrementando il byte del valore opportuno.
Nelle condizioni normali la MAPPA VIDEO si trova nelle 1000 locazioni che vanno dal byte 1024 al byte 2023; la MAPPA COLORI si trova nelle 1000 locazioni che vanno dal byte 55796 al byte 56795. Le posizioni di queste due mappe corrispondono ordinatamente alle posizioni del video partendo dall'angolo in alto a sinistra e avanzando carattere per carattere lungo le righe. Affinchè un carattere compaia in una posizione del video, devono essere verificate le seguenti condizioni:
.. nella posizione della MAPPA VIDEO deve essere scritto il codice del carattere in D/CODE;
.. nella posizione corrispondente della MAPPA COLORE deve essere scritto il codice del colore per il carattere, diverso dal colore dello sfondo.
Il sistema va a prelevare dalla tabella descrittiva per punti dei caratteri, tenendo conto del SET selezionato, i bytes che descrivono il carattere e fa comparire i punti che disegnano il carattere sul video. Il D/CODE fa da puntatore per prelevare il carattere nella MAPPA CARATTERI.
La risoluzione del video è in caratteri di 25 righe di 40 caratteri ciascuna; passando ai punti, 8 x 8 per ogni carattere, si ha: (8 x 25) x (8 x 40) = 64000.

Al momento dell'inizializzazione la MAPPA COLORE contiene il codice del colore dello sfondo in tutte le posizioni.

Per stampare sul video si può procedere nei seguenti modi:

1) Usare il comando PRINT seguito dal nome di una variabile numerica o stringa oppure da una costante numerica o stringa; viene evidenziato, carattere per carattere, il contenuto della variabile o la costante, a partire dalla posizione attuale del cursore, nel colore del cursore.

2) Usare il comando PRINT seguito dalla funzione CHR$(x), dove x è il codice ASCII di un carattere stampabile; la stampa avviene con le stesse modalità del punto 1).

3) Scrivere con il comando POKE in una posizione della MAPPA VIDEO il codice D/CODE del carattere e nella corrispondente posizione della MAPPA COLORI il codice del colore desiderato per il carattere (diverso dal codice del colore dello sfondo).

Nei casi 1) e 2) pensano i programmi del sistema a fare quello che fai tu nel caso 3).

Il programma che segue produce una tabella nella quale sono riportati i caratteri stampabili, cioè quelli il cui codice ASCII va da 32 a 127 e da 160 a 255.

```
0 REM CARATTERI E CODICI NEL CBM 64
1 PRINT"{CLR}{DOWN}{DOWN}{DOWN}            PROGRAMMA CARATTERI"
2 DIMA(96),D(96),C$(96),I$(96),EA$(96)
3 DIMBA$(96),H$(256),K$(16),IN(96)
4 REM MAPPA VIDEO  DA 1024 A 2023
5 REM MAPPA COLORI DA 55296 A 56295
6 MV=1024:MC=55296
7 REM PUNT. MAPPA VIDEO E MAPPA COLORI
8 GOSUB5000:PRINTCHR$(142):SW=0
9 REM SET MAIUSCOLO/GRAFICO
10 PRINT"{DOWN}{DOWN}{DOWN}          VUOI USARE LA PRINTER S/N ";
11 INPUTR$:PRINT
12 REM STAMPA CARATTERI STAMPABILI
13 REM CODICI ASCII E D/CODE CORRISPONDENTI
14 REM CODICI ASCII DA 32 A 63
15 K1=32:K2=63:PRINT"{CLR}";:FORK=K1TOK2
16 PRINTCHR$(K);:NEXTK:PRINT
20 I=0:FORK=K1TOK2
21 POKEMV+32+I,K+128:POKEMC+32+I,1
22 I=I+1:NEXTK
30 GOSUB1100:GOSUB1200:GOSUB1300
40 REM CODICI ASCII DA 64 A 127
45 K1=64:K2=127:GOSUB1000
50 GOSUB1100:GOSUB1200:GOSUB1300
55 REM CODICI ASCII DA 160 A 255
60 K1=160:K2=255:GOSUB1000
65 GOSUB1100:GOSUB1200:GOSUB1300
70 REMSET MAIUSCOLO/MINUSCOLO
80 IFSW=0THENSW=1:PRINTCHR$(14):GOTO14
999 STOP
1000 PRINT"{CLR}";:FORK=K1TOK2:PRINTCHR$(K);
1001 NEXTK:PRINTCHR$(18);:FORK=K1TOK2
```

```
1004 PRINTCHR$(K);
1005 NEXTK:PRINTCHR$(146):RETURN
1100 I=1:FORK=K1TOK2:A(I)=K
1101 D(I)=PEEK(MV-1+I)
1103 IN(I)=PEEK(MV-1+I+K2-K1+1)
1105 C$(I)=CHR$(K):I$(I)="▨"+C$(I)+"▀"
1110 EA$(I)=H$(K):H1$=LEFT$(EA$(I),1)
1111 H2$=RIGHT$(EA$(I),1)
1115 H1=ASC(H1$):H2=ASC(H2$)
1116 IFH1<=57THENH1=H1-48:GOTO1118
1117 H1=H1-55
1118 IFH2<=57THENH2=H2-48:GOTO1125
1120 H2=H2-55
1125 BA$(I)=K$(H1)+K$(H2)
1130 I=I+1:NEXTK:RETURN
1200 PRINT"▯":GOSUB2000:I=1
1203 FORK=K1TOK2
1204 IFK<>34THEN1208
1205 PRINTTAB(2)CHR$(34)CHR$(34)"▌ "TAB(6);
1206 PRINT"▨"CHR$(34)CHR$(34)"▀▌ ";
1207 GOTO1210
1208 PRINTTAB(2)C$(I)TAB(6)I$(I);
1210 PRINTTAB(11)A(I)TAB(16)EA$(I);
1213 PRINTTAB(20)BA$(I);
1215 PRINTTAB(30)D(I)TAB(34)IN(I)
1220 FORJ=0TO500:NEXTJ:I=I+1:NEXTK
1221 RETURN
1300 IFR$="N"THENRETURN
1303 OPEN4,4:CMD4
1305 IFSW=0THENPRINTCHR$(145);:GOTO1307
1306 PRINTCHR$(17);
1307 GOSUB2000:I=1:FORK=K1TOK2
1310 PRINTCHR$(16)CHR$(48)CHR$(50)C$(I);
1311 PRINTCHR$(16)CHR$(48)CHR$(54)I$(I);
1315 PRINTCHR$(16)CHR$(48)CHR$(59)A(I);
1317 PRINTCHR$(16)CHR$(49)CHR$(55)EA$(I);
1318 PRINTCHR$(16)CHR$(50)CHR$(50)BA$(I);
1319 PRINTCHR$(16);CHR$(51)CHR$(48)D(I);
1321 PRINTCHR$(16)CHR$(51)CHR$(53)IN(I)
1323 I=I+1:NEXTK
1350 PRINTCHR$(145);:PRINT#4:CLOSE4
1351 RETURN
2000 PRINT"CORRISP.  CARATTERI, CODICI ";
```

```
2001 PRINT"ASCII, D/CODE"
2005 PRINT"I CODICI ASCII SONO IN ";
2006 PRINT"DEC., ESADEC. E BIN."
2010 IFSW=1THEN2015
2011 PRINT"SET MAIUSCOLO/GRAFICO":GOTO2020
2015 PRINT"SET MAIUSCOLO/MINUSCOLO"
2020 PRINT
2025 PRINT"CARATT.                   ";
2026 PRINT"ASCII              D/CODE"
2030 PRINT"DIR. INV.  DEC. ";
2031 PRINT"HEX.  BIN.        DEC."
2035 PRINT:RETURN
5000 DATA"000102030405060708090A0B0C0D0E0F"
5001 DATA"101112131415161718191A1B1C1D1E1F"
5003 DATA"202122232425262728292A2B2C2D2E2F"
5004 DATA"303132333435363738393A3B3C3D3E3F"
5005 DATA"404142434445464748494A4B4C4D4E4F"
5006 DATA"505152535455565758595A5B5C5D5E5F"
5007 DATA"606162636465666768696A6B6C6D6E6F"
5008 DATA"707172737475767778797A7B7C7D7E7F"
5009 DATA"808182838485868788898A8B8C8D8E8F"
5010 DATA"909192939495969798999A9B9C9D9E9F"
5011 DATA"A0A1A2A3A4A5A6A7A8A9AAABACADAEAF"
5012 DATA"B0B1B2B3B4B5B6B7B8B9BABBBCBDBEBF"
5013 DATA"C0C1C2C3C4C5C6C7C8C9CACBCCCDCECF"
5014 DATA"D0D1D2D3D4D5D6D7D8D9DADBDCDDDEDF"
5015 DATA"E0E1E2E3E4E5E6E7E8E9EAEBECEDEEEF"
5016 DATA"F0F1F2F3F4F5F6F7F8F9FAFBFCFDFEFF"
5017 K=0:FORJ=0TO15:READA$:FORL=1TO16
5018 H$(K)=MID$(A$,L*2-1,2)
5019 K=K+1:NEXTL:NEXTJ
5020 DATA"00000001001000110100010101100111"
5021 DATA"10001001101010111100110111101111"
5022 K=0:FORJ=0TO1:READA$:FORL=0TO7
5023 K$(K)=MID$(A$,(L+1)*4-3,4):K=K+1
5024 NEXTL:NEXTJ
5025 RETURN
```

Nelle prime due coŀonne della tabella sono riportati i caratteri come appaiono sulla stampante in modo diretto e inverso, nelle tre colonne seguenti sono riportati i codici ASCII in decimale, esadecimale e binario, e nelle ultime due colonne sono riportati i codici decimali D/CODE usati nella mappa video per il carattere diretto e inverso. Il codice del carattere inverso si ottiene aggiungendo 128 al codice del carattere diretto.

RISULTATI DEL PROGRAMMA PRECEDENTE

```
CARATT.
CORRISP.  CARATTERI, CODICI ASCII, D/CODE
I CODICI ASCII SONO IN DEC., ESADEC. E BIN.
SET MAIUSCOLO/GRAFICO
```

CARATT.		ASCII			D/CODE	
DIR.	INV.	DEC.	HEX.	BIN.	DEC.	
		32	20	00100000	32	160
!		33	21	00100001	33	161
"		34	22	00100010	34	162
#		35	23	00100011	35	163
$		36	24	00100100	36	164
%		37	25	00100101	37	165
&		38	26	00100110	38	166
'		39	27	00100111	39	167
(		40	28	00101000	40	168
)		41	29	00101001	41	169
*		42	2A	00101010	42	170
+		43	2B	00101011	43	171
,		44	2C	00101100	44	172
-		45	2D	00101101	45	173
.		46	2E	00101110	46	174
/		47	2F	00101111	47	175
0		48	30	00110000	48	176
1		49	31	00110001	49	177
2		50	32	00110010	50	178
3		51	33	00110011	51	179
4		52	34	00110100	52	180
5		53	35	00110101	53	181
6		54	36	00110110	54	182
7		55	37	00110111	55	183
8		56	38	00111000	56	184
9		57	39	00111001	57	185
:		58	3A	00111010	58	186
;		59	3B	00111011	59	187
<		60	3C	00111100	60	188
=		61	3D	00111101	61	189
>		62	3E	00111110	62	190
?		63	3F	00111111	63	191

CORRISP. CARATTERI, CODICI ASCII, D/CODE
I CODICI ASCII SONO IN DEC., ESADEC. E BIN.
SET MAIUSCOLO/GRAFICO

CARATT. DIR.	INV.	ASCII DEC.	HEX.	BIN.		D/CODE DEC.
		99	63	01100011	67	195
		100	64	01100100	68	196
		101	65	01100101	69	197
		102	66	01100110	70	198
|		103	67	01100111	71	199
|		104	68	01101000	72	200
		105	69	01101001	73	201
		106	6A	01101010	74	202
		107	6B	01101011	75	203
L		108	6C	01101100	76	204
		109	6D	01101101	77	205
		110	6E	01101110	78	206
		111	6F	01101111	79	207
		112	70	01110000	80	208
		113	71	01110001	81	209
		114	72	01110010	82	210
		115	73	01110011	83	211
|		116	74	01110100	84	212
		117	75	01110101	85	213
✕		118	76	01110110	86	214
O		119	77	01110111	87	215
		120	78	01111000	88	216
|		121	79	01111001	89	217
		122	7A	01111010	90	218
+		123	7B	01111011	91	219
		124	7C	01111100	92	220
|		125	7D	01111101	93	221
π		126	7E	01111110	94	222
		127	7F	01111111	95	223
		160	A0	10100000	96	224
		161	A1	10100001	97	225
		162	A2	10100010	98	226
		163	A3	10100011	99	227
		164	A4	10100100	100	228
|		165	A5	10100101	101	229
		166	A6	10100110	102	230

CORRISP. CARATTERI, CODICI ASCII, D/CODE
I CODICI ASCII SONO IN DEC., ESADEC. E BIN.
SET MAIUSCOLO/GRAFICO

CARATT.		ASCII			D/CODE	
DIR.	INV.	DEC.	HEX.	BIN.	DEC.	
		167	A7	10100111	103	231
		168	A8	10101000	104	232
		169	A9	10101001	105	233
		170	AA	10101010	106	234
		171	AB	10101011	107	235
		172	AC	10101100	108	236
		173	AD	10101101	109	237
		174	AE	10101110	110	238
		175	AF	10101111	111	239
		176	B0	10110000	112	240
		177	B1	10110001	113	241
		178	B2	10110010	114	242
		179	B3	10110011	115	243
		180	B4	10110100	116	244
		181	B5	10110101	117	245
		182	B6	10110110	118	246
		183	B7	10110111	119	247
		184	B8	10111000	120	248
		185	B9	10111001	121	249
		186	BA	10111010	122	250
		187	BB	10111011	123	251
		188	BC	10111100	124	252
		189	BD	10111101	125	253
		190	BE	10111110	126	254
		191	BF	10111111	127	255
		192	C0	11000000	64	192
		193	C1	11000001	65	193
		194	C2	11000010	66	194
		195	C3	11000011	67	195
		196	C4	11000100	68	196
		197	C5	11000101	69	197
		198	C6	11000110	70	198
		199	C7	11000111	71	199
		200	C8	11001000	72	200
		201	C9	11001001	73	201
		202	CA	11001010	74	202

CORRISP. CARATTERI, CODICI ASCII, D/CODE
I CODICI ASCII SONO IN DEC., ESADEC. E BIN.
SET MAIUSCOLO/GRAFICO

CARATT.		ASCII				D/CODE
DIR.	INV.	DEC.	HEX.	BIN.		DEC.
		203	CB	11001011	75	203
		204	CC	11001100	76	204
		205	CD	11001101	77	205
		206	CE	11001110	78	206
		207	CF	11001111	79	207
		208	D0	11010000	80	208
		209	D1	11010001	81	209
		210	D2	11010010	82	210
		211	D3	11010011	83	211
		212	D4	11010100	84	212
		213	D5	11010101	85	213
		214	D6	11010110	86	214
		215	D7	11010111	87	215
		216	D8	11011000	88	216
		217	D9	11011001	89	217
		218	DA	11011010	90	218
		219	DB	11011011	91	219
		220	DC	11011100	92	220
		221	DD	11011101	93	221
		222	DE	11011110	94	222
		223	DF	11011111	95	223
		224	E0	11100000	96	224
		225	E1	11100001	97	225
		226	E2	11100010	98	226
		227	E3	11100011	99	227
		228	E4	11100100	100	228
		229	E5	11100101	101	229
		230	E6	11100110	102	230
		231	E7	11100111	103	231
		232	E8	11101000	104	232
		233	E9	11101001	105	233
		234	EA	11101010	106	234
		235	EB	11101011	107	235
		236	EC	11101100	108	236
		237	ED	11101101	109	237

CORRISP. CARATTERI, CODICI ASCII, D/CODE
I CODICI ASCII SONO IN DEC., ESADEC. E BIN.
SET MAIUSCOLO/GRAFICO

CARATT. DIR.	INV.	ASCII DEC.	HEX.	BIN.		D/CODE DEC.
┐		238	EE	11101110	110	238
		239	EF	11101111	111	239
┌		240	F0	11110000	112	240
┴		241	F1	11110001	113	241
┬		242	F2	11110010	114	242
┤		243	F3	11110011	115	243
|		244	F4	11110100	116	244
		245	F5	11110101	117	245
		246	F6	11110110	118	246
		247	F7	11110111	119	247
		248	F8	11111000	120	248
		249	F9	11111001	121	249
┘		250	FA	11111010	122	250
		251	FB	11111011	123	251
		252	FC	11111100	124	252
┘		253	FD	11111101	125	253
		254	FE	11111110	126	254
π		255	FF	11111111	94	222

corrisp. caratteri, codici ascii, d/code
i codici ascii sono in dec., esadec. e bin.
set maiuscolo/minuscolo

caratt. dir.	inv.	ascii dec.	hex.	bin.		d/code dec.
		32	20	00100000	32	160
!		33	21	00100001	33	161
"		34	22	00100010	34	162
#		35	23	00100011	35	163
$		36	24	00100100	36	164
%		37	25	00100101	37	165
&		38	26	00100110	38	166
'		39	27	00100111	39	167

corrisp. caratteri, codici ascii, d/code
i codici ascii sono in dec., esadec. e bin.
set maiuscolo/minuscolo

caratt. dir.	inv.	ascii dec.	hex.	bin.	d/code dec.	
(	(	40	28	00101000	40	168
)	)	41	29	00101001	41	169
*	*	42	2a	00101010	42	170
+	+	43	2b	00101011	43	171
,	,	44	2c	00101100	44	172
-	-	45	2d	00101101	45	173
.	.	46	2e	00101110	46	174
/	/	47	2f	00101111	47	175
0	0	48	30	00110000	48	176
1	1	49	31	00110001	49	177
2	2	50	32	00110010	50	178
3	3	51	33	00110011	51	179
4	4	52	34	00110100	52	180
5	5	53	35	00110101	53	181
6	6	54	36	00110110	54	182
7	7	55	37	00110111	55	183
8	8	56	38	00111000	56	184
9	9	57	39	00111001	57	185
:	:	58	3a	00111010	58	186
;	;	59	3b	00111011	59	187
<	<	60	3c	00111100	60	188
=	=	61	3d	00111101	61	189
>	>	62	3e	00111110	62	190
?	?	63	3f	00111111	63	191
@	@	64	40	01000000	0	128
a	a	65	41	01000001	1	129
b	b	66	42	01000010	2	130
c	c	67	43	01000011	3	131
d	d	68	44	01000100	4	132
e	e	69	45	01000101	5	133
f	f	70	46	01000110	6	134
g	g	71	47	01000111	7	135
h	h	72	48	01001000	8	136
i	i	73	49	01001001	9	137
j	j	74	4a	01001010	10	138
k	k	75	4b	01001011	11	139

corrisp. caratteri, codici ascii, d/code
i codici ascii sono in dec., esadec. e bin.
set maiuscolo/minuscolo

caratt. dir.	caratt. inv.	ascii dec.	ascii hex.	ascii bin.	d/code dec.	
l		76	4c	01001100	12	140
m		77	4d	01001101	13	141
n		78	4e	01001110	14	142
o		79	4f	01001111	15	143
p		80	50	01010000	16	144
q		81	51	01010001	17	145
r		82	52	01010010	18	146
s		83	53	01010011	19	147
t		84	54	01010100	20	148
u		85	55	01010101	21	149
v		86	56	01010110	22	150
w		87	57	01010111	23	151
x		88	58	01011000	24	152
y		89	59	01011001	25	153
z		90	5a	01011010	26	154
[		91	5b	01011011	27	155
£		92	5c	01011100	28	156
]		93	5d	01011101	29	157
↑		94	5e	01011110	30	158
←		95	5f	01011111	31	159
—		96	60	01100000	64	192
A		97	61	01100001	65	193
B		98	62	01100010	66	194
C		99	63	01100011	67	195
D		100	64	01100100	68	196
E		101	65	01100101	69	197
F		102	66	01100110	70	198
G		103	67	01100111	71	199
H		104	68	01101000	72	200
I		105	69	01101001	73	201
J		106	6a	01101010	74	202
K		107	6b	01101011	75	203
L		108	6c	01101100	76	204
M		109	6d	01101101	77	205
N		110	6e	01101110	78	206
O		111	6f	01101111	79	207

corrisp. caratteri, codici ascii, d/code
i codici ascii sono in dec., esadec. e bin.
set maiuscolo/minuscolo

caratt. dir.	caratt. inv.	ascii dec.	ascii hex.	ascii bin.	d/code dec.	
P		112	70	01110000	80	208
Q		113	71	01110001	81	209
R		114	72	01110010	82	210
S		115	73	01110011	83	211
T		116	74	01110100	84	212
U		117	75	01110101	85	213
V		118	76	01110110	86	214
W		119	77	01110111	87	215
X		120	78	01111000	88	216
Y		121	79	01111001	89	217
Z		122	7a	01111010	90	218
+		123	7b	01111011	91	219
		124	7c	01111100	92	220
|		125	7d	01111101	93	221
		126	7e	01111110	94	222
		127	7f	01111111	95	223
		160	a0	10100000	96	224
		161	a1	10100001	97	225
		162	a2	10100010	98	226
		163	a3	10100011	99	227
		164	a4	10100100	100	228
		165	a5	10100101	101	229
		166	a6	10100110	102	230
		167	a7	10100111	103	231
		168	a8	10101000	104	232
		169	a9	10101001	105	233
		170	aa	10101010	106	234
		171	ab	10101011	107	235
		172	ac	10101100	108	236
		173	ad	10101101	109	237
		174	ae	10101110	110	238
		175	af	10101111	111	239
		176	b0	10110000	112	240
		177	b1	10110001	113	241
		178	b2	10110010	114	242
		179	b3	10110011	115	243

corrisp. caratteri, codici ascii, d/code
i codici ascii sono in dec., esadec. e bin.
set maiuscolo/minuscolo

caratt.		ascii			d/code	
dir.	inv.	dec.	hex.	bin.	dec.	
|	▌	180	b4	10110100	116	244
▌	▌	181	b5	10110101	117	245
▐	▌	182	b6	10110110	118	246
‾	■	183	b7	10110111	119	247
▬	■	184	b8	10111000	120	248
▂	▀	185	b9	10111001	121	249
✓	■	186	ba	10111010	122	250
▖	▜	187	bb	10111011	123	251
▝	▙	188	bc	10111100	124	252
┘	■	189	bd	10111101	125	253
▘	▟	190	be	10111110	126	254
▚	▞	191	bf	10111111	127	255
—	■	192	c0	11000000	64	192
A	A	193	c1	11000001	65	193
B	B	194	c2	11000010	66	194
C	C	195	c3	11000011	67	195
D	D	196	c4	11000100	68	196
E	E	197	c5	11000101	69	197
F	F	198	c6	11000110	70	198
G	G	199	c7	11000111	71	199
H	H	200	c8	11001000	72	200
I	I	201	c9	11001001	73	201
J	J	202	ca	11001010	74	202
K	K	203	cb	11001011	75	203
L	L	204	cc	11001100	76	204
M	M	205	cd	11001101	77	205
N	N	206	ce	11001110	78	206
O	O	207	cf	11001111	79	207
P	P	208	d0	11010000	80	208
Q	Q	209	d1	11010001	81	209
R	R	210	d2	11010010	82	210
S	S	211	d3	11010011	83	211
T	T	212	d4	11010100	84	212
U	U	213	d5	11010101	85	213
V	V	214	d6	11010110	86	214

corrisp. caratteri, codici ascii, d/code
i codici ascii sono in dec., esadec. e bin.
set maiuscolo/minuscolo

caratt.		ascii				d/code
dir.	inv.	dec.	hex.	bin.		dec.
W		215	d7	11010111	87	215
X		216	d8	11011000	88	216
Y		217	d9	11011001	89	217
Z		218	da	11011010	90	218
+		219	db	11011011	91	219
		220	dc	11011100	92	220
|		221	dd	11011101	93	221
		222	de	11011110	94	222
		223	df	11011111	95	223
		224	e0	11100000	96	224
		225	e1	11100001	97	225
		226	e2	11100010	98	226
		227	e3	11100011	99	227
		228	e4	11100100	100	228
		229	e5	11100101	101	229
		230	e6	11100110	102	230
		231	e7	11100111	103	231
		232	e8	11101000	104	232
		233	e9	11101001	105	233
		234	ea	11101010	106	234
		235	eb	11101011	107	235
		236	ec	11101100	108	236
		237	ed	11101101	109	237
		238	ee	11101110	110	238
		239	ef	11101111	111	239
		240	f0	11110000	112	240
		241	f1	11110001	113	241
		242	f2	11110010	114	242
		243	f3	11110011	115	243
		244	f4	11110100	116	244
		245	f5	11110101	117	245
		246	f6	11110110	118	246
		247	f7	11110111	119	247
		248	f8	11111000	120	248
		249	f9	11111001	121	249

```
corrisp.  caratteri, codici ascii, d/code
i codici ascii sono in dec., esadec. e bin.
set maiuscolo/minuscolo

carat t.              ascii              d/code
dir.  inv.   dec.  hex.   bin.           dec.

                250   fa    11111010 122    250
                251   fb    11111011 123    251
                252   fc    11111100 124    252
                253   fd    11111101 125    253
                254   fe    11111110 126    254
                255   ff    11111111 94     222
```

Il programma usa questa tecnica:

.. pulisce il video e stampa a partire dalla prima posizione con la funzione CHR$(k) un gruppo di caratteri (sottoprogramma in 1000);

.. per il gruppo di caratteri di codice ASCII compreso tra 32 e 63 abbiamo dovuto usare degli accorgimenti per impedire che il carattere "aperte virgolette" disturbasse la stampa;

.. va a leggere con la funzione PEEK (sottoprogramma 1100) dalle posizioni del video, quello che ha stampato e ricava il D/CODE corrispondente ad ogni valore di K usato nella funzione CHR$; inoltre preleva dal vettore H$ il valore esadecimale del codice ASCII e dal vettore K$ il valore binario corrispondente;

.. stampa sul video (sottoprogramma 1200) la tabella ricavata, usando la funzione TAB per ottenere gli incolonnamenti;

.. stampa sulla stampante (sottoprogramma 1300), se abilitata, la tabella usando, invece della funzione TAB, il codice di controllo per gli incolonnamenti: CHR$(16) seguito dai due CHR$ necessari per dare la posizione di stampa cifra per cifra;

.. i 256 codici ASCII in esadecimale sono caricati nel sottoprogramma in 5000 nel vettore H$, ricavandoli dai DATA delle linee 5000-5016, e i 16 valori binari corrispondenti alle 16 cifre esadecimali sono memorizzati nel vettore K$, ricavandoli dai DATA delle linee 5020 e 5021. Nel programma precedente, che stampa il disegno dei caratteri usando gli asterischi, i valori binari sono ottenuti con un procedimento diverso da quello seguito qui. Analogamente i codici ASCII in

esadecimale si possono ottenere con altre sequenze di istruzioni; puoi provare a modificare questo programma;

.. terminata la stampa del set di caratteri maiuscolo/grafico, si passa al set maiuscolo/minuscolo e si esegue nuovamente il programma;

.. per poter ottenere sulla stampante il set maiuscole/minuscole si deve abilitare la stampa del set stesso usando il codice di controllo 17 (PRINT CHR$(17)), mentre per tornare al set maiuscolo/grafico si deve usare il codice di controllo 145 (PRINT CHR$(145));

.. osservando il listato su stampante dei risultati, prima riportato, puoi vedere una irregolarità alla linea che riguarda il codice ASCII 34; questo codice crea un pò di problemi, dato che quando viene stampato ha l'effetto di aprire le virgolette, e questo effetto si trascina e fa stampare anche il carattere di REVERS ON e altri caratteri prima delle altre virgolette. Per risolvere lo stesso problema sul video si è usata una tecnica particolare che puoi rilevare dalle linee 1204-1207 del listato del programma. Sul video si può tornare indietro e cancellare, sulla carta stampata no.

Se osservi con attenzione i risultati stampati dal programma precedente puoi vedere che:

.. i codici ASCII da 192 a 223 danno luogo agli stessi caratteri dei codici da 96 a 127;

.. i codici ASCII da 224 a 254 danno luogo agli stessi caratteri dei codici da 160 a 190;

.. il codice ASCII 255 dà luogo allo stessso carattere del codice 126;

.. nei due set si hanno caratteri identici per i seguenti gruppi di codici:

ASCII	D/CODE
32 - 64	32 - 63 e 0
91 - 96	27 - 64
123 - 125	91 - 94
160 - 168	96 - 104
170 - 185	106 - 121
187 - 191	123 - 127

7.5 CODICI DI CONTROLLO

Ci occupiamo ora dei CODICI DI CONTROLLO, cioè di quei codici che in ASCII corrispondono nei due set di caratteri ai due gruppi di numeri, da 0 a 31 e da 128 a 159. Alcuni di questi codici non hanno significato, e dato che non vengono riconosciuti come caratteri stampabili possono dar luogo ad errori. Questi caratteri non hanno un corrispondente in D/CODE.

I codici di controllo che passiamo ad elencare vengono attivati dal comando PRINT e possono essere passati al sistema come CHR$ o all'interno di una stringa delimitata dalle virgolette (apici, di codice ASCII 34).

CODICE ASCII	AZIONE
5	colore bianco
8	disabilita l'effetto della pressione contemporanea dei tasti SHIFT e COMMODORE se diretto alla stampante attiva il modo grafico per punti
9	abilita l'effetto della pressione cont=temporanea dei tasti SHIFT e COMMODORE
10	manda un LINE FEED alla stampante
13	corrisponde alla pressione del tasto RETURN sulla stampante produce anche line feed
14	fa passare al SET minuscolo/grafico se diretto alla stampante fa passare a caratteri di doppia ampiezza
15	se diretto alla stampante disabilita l'effetto del codice 14
16	se diretto alla stampante provoca lo spostamento della posizione
17	produce l'effetto del tasto CRSR GIU' se diretto alla stampante attiva il set di caratteri maiuscolo/minuscolo
18	attiva il modo REVERSE ON anche sulla stampante
19	manda il cursore nell'angolo in alto a sinistra
20	attiva la funzione di DELETE
26	se diretto alla stampante fa ripetere i caratteri grafici

27	se diretto alla stampante sposta a una posizione punto specificata
28	colore rosso
29	produce l'effetto del tasto CRSR DESTRA
30	colore verde
31	colore blu
129	colore arancione
133	tasto funzione F1
134	tasto funzione F3
135	tasto funzione F5
136	tasto funzione F7
137	tasto funzione F2
138	tasto funzione F4
139	tasto funzione F6
140	tasto funzione F8
141	produce lo stesso effetto della pressione contemporanea dei tasti SHIFT e RETURN
142	attiva il set maiuscolo/grafico
144	colore nero
145	produce l'effetto del tasto CRSR SU se diretto alla stampante attiva il set maiuscolo/grafico
146	annulla l'effetto di RVS ON corrisponde quindi a RVS OFF anche se diretto alla stampante
147	pulisce il video e manda il cursore nell'angolo in alto a sinistra
148	attiva la funzione di INSERT
149	colore bruno
150	color rosso chiaro
151	colore grigio chiaro
152	colore grigio scuro
153	colore verde chiaro
154	colore blu chiaro
155	colore grigio molto scuro
156	colore porpora
157	produce l'effetto del tasto CRSR SINISTRA
158	colore giallo
159	colore ciano

Come vedi alcuni caratteri agiscono diversamente se diretti con il comando PRINT al video (vedi Capitolo 3) o alla stampante (vedi Capitolo 5).

Quando ai tasti funzione viene assegnato un compito specifico, il corrispondente codice di controllo lo fa eseguire.

7.6 CODIFICA PAROLE CHIAVE - TOKENS

Le parole chiave del linguaggio Basic si scrivono sulla tastiera carattere per carattere, ma, salvo poche eccezioni, vengono codificate in un byte. Questi codici speciali prendono il nome di TOKENS. Essi coprono l'intervallo dei numeri da 128 a 202, ma non possono essere scambiati con i normali caratteri corrispondenti a questi numeri, dato che vengono interpretati come parole chiave nel contesto nel quale sono presi in esame.

Inoltre, molte parole chiave possono essere digitate sulla tastiera in forma abbreviata; il sistema le riconosce, ma produce sul video una rappresentazione simbolica abbreviata. In memoria vengono memorizzate allo stesso modo di quando si digitano per intero, e, in fase di lista del programma compaiono in forma completa.

Il programma TOKENS che segue produce una tabella, il cui contenuto è il seguente:

.. col. 1) COD.: codici decimali delle parole chiave;

.. col. 2) DES.: parola, o parole chiave, o simboli;

.. col. 3) ABBR.: tasti da digitare sulla tastiera per abbreviare; devi intendere il primo, o i primi due tasti e poi il tasto SHIFT insieme al tasto che segue. Se non esiste abbreviazione, la posizione è in bianco;

.. col. 4) VIDEO: quello che compare sul video per effetto della digitazione abbreviata (di quella normale se non esiste abbreviazione).

```
1 REM TOKENS
5 OPEN4,4:CMD4:GOSUB150:J=128
7 PRINTCHR$(14);"              T O K E N S":PRINT
8 PRINT" COD. DES.      ABBR.      VIDEO":PRINT
9 N=1  :L$=""
10 FORK=41118TO41370
15 L=PEEK(K)
20 IFL<=127THENL$=L$+CHR$(L):NEXTK
25 L=L-128:L$=L$+CHR$(L)
27 L$=LEFT$(L$+S$,7)
```

```
30 PRINTJ;L$;" ";
35 IFA1$(N)<>S$THEN40
37 PRINTS$;"  ";L$:GOTO50
40 PRINTA1$(N);"  ";A2$(N)
50 N=N+1:J=J+1:L$="":NEXTK:PRINTCHR$(15)
100 PRINT#4:CLOSE4:STOP
150 DATA"E N/F O N E D A♣I N/    D I R E"
151 DATA" L S G O "
155 DATA"R U,    RES♠GOS♠RET!    S T!   "
156 DATA" W A♣L O "
160 DATA"S A♣V E D E P O P R_? ? C O L I"
161 DATA" C LLC M "
165 DATA"S Y O P CLO G E     T A♣      "
166 DATA" S P T H !"
170 DATA"N O STE                        R N"
171 DATA"/           "
175 DATA"         S G!    A BIU S♠F R_    "
176 DATA" S Q●R N/"
180 DATA"     E X♣    S I     A TIP E    "
181 DATA" STR_V A♣"
185 DATA"A S♠C H ILEF_R I M I "
208 DIMA1$(75),A2$(75):S$="          "
209 L=1:FORK=0TO6:READM$:READN$:M$=M$+N$
210 FORJ=0TO9:A1$(L)=MID$(M$,J*4+1,4)
211 L=L+1:NEXTJ:NEXTK
212 READM$:FORJ=0TO4:A1$(L)=MID$(M$,J*4+1,4)
213 L=L+1:NEXTJ:FORK=1TO75
214 A2$(K)=LEFT$(A1$(K),2)
215 A2$(K)=A2$(K)+"  "+RIGHT$(A1$(K),1)
217 IFLEFT$(A1$(K),1)<>"?"THEN220
218 A1$(K)="    ?    "
219 A2$(K)="  ?  ":GOTO230
220 IFA1$(K)="    "THENA1$(K)=S$:GOTO230
221 N$=LEFT$(A1$(K),2)+" SHIFT "
223 A1$(K)=N$+MID$(A1$(K),3,1)
230 NEXTK
250 RETURN
```

Il programma, dopo aver aperto la stampante alla linea 5, va ad eseguire il sottoprogramma in 150. In esso vengono preparati i due vettori A1$ e A2$ per contenere rispettivamente le colonne 3 e 4 della tabella. I dati per questi vettori sono stati passati con delle istruzioni DATA. Puoi analizzare il sottoprogramma e vedrai

che abbiamo dovuto usare degli accorgimenti per quei codici che non ammettono abbreviazioni. Dopo l'esecuzione del sottoprogramma, viene inizializzato J a 128, codice della parola chiave END.
Alla linea 10 inizia un ciclo FOR per K che varia da 41118 a 41370; questi due numeri sono gli indirizzi che delimitano una zona di memoria ROM del Basic dove sono memorizzate le parole chiave carattere per carattere. La tecnica di memorizzazione è la seguente: una parola chiave viene memorizzata carattere per carattere con il codice ASCII corrispondente; per segnalare la sua fine, viene sommato 128 (80H) al codice dell'ultimo carattere. Nel programma si analizzano i codici letti e quando un codice supera 127 si considera conclusa la parola. Il sistema usa i codici delle parole chiave per puntare alle parole stesse nella tabella ora menzionata; può usare il sistema di creare un puntatore che inizia da 0, se sottrae 128 al codice, oppure che inizia da 1, se sottrae 127. Per il momento non siamo andati a vedere come il sistema agisce, ma quello che importa e che tu ti impadronisca di questi tipi di tecniche di programmazione.

RISULTATI PROGRAMMA TOKENS

T O K E N S

COD.	DES.	ABBR.			VIDEO	
128	END	E	SHIFT	N	E	╱
129	FOR	F	SHIFT	O	F	┌
130	NEXT	N	SHIFT	E	N	─
131	DATA	D	SHIFT	A	D	♠
132	INPUT#	I	SHIFT	N	I	╱
133	INPUT				INPUT	
134	DIM	D	SHIFT	I	D	╲
135	READ	R	SHIFT	E	R	─
136	LET	L	SHIFT	E	L	─
137	GOTO	G	SHIFT	O	G	┌
138	RUN	R	SHIFT	U	R	╭
139	IF				IF	
140	RESTORE	RE	SHIFT	S	RE	♥
141	GOSUB	GO	SHIFT	S	GO	♥
142	RETURN	RE	SHIFT	T	RE	│
143	REM				REM	
144	STOP	S	SHIFT	T	S	│
145	ON				ON	
146	WAIT	W	SHIFT	A	W	♠
147	LOAD	L	SHIFT	O	L	┌
148	SAVE	S	SHIFT	A	S	♠
149	VERIFY	V	SHIFT	E	V	─
150	DEF	D	SHIFT	E	D	─
151	POKE	P	SHIFT	O	P	┌
152	PRINT#	P	SHIFT	R	P	▁

153	PRINT		?		?
154	CONT	C	SHIFT	O	C ┌
155	LIST	L	SHIFT	I	L ╮
156	CLR	C	SHIFT	L	C └
157	CMD	C	SHIFT	M	C ╲
158	SYS	S	SHIFT	Y	S │
159	OPEN	O	SHIFT	P	O ┐
160	CLOSE	CL	SHIFT	O	CL ┌
161	GET	G	SHIFT	E	G ‾
162	NEW				NEW
163	TAB(	T	SHIFT	A	T ♠
164	TO				TO
165	FN				FN
166	SPC(	S	SHIFT	P	S ┐
167	THEN	T	SHIFT	H	T │
168	NOT	N	SHIFT	O	N ┌
169	STEP	ST	SHIFT	E	ST ‾
170	+				+
171	−				−
172	*				*
173	/				/
174	↑				↑
175	AND	A	SHIFT	N	A ╱
176	OR				OR
177	>				>
178	=				=
179	<				<
180	SGN	S	SHIFT	G	S │
181	INT				INT
182	ABS	A	SHIFT	B	A │
183	USR	U	SHIFT	S	U ♥
184	FRE	F	SHIFT	R	F ▁
185	POS				POS
186	SQR	S	SHIFT	Q	S ●
187	RND	R	SHIFT	N	R ╱
188	LOG				LOG
189	EXP	E	SHIFT	X	E ♣
190	COS				COS
191	SIN	S	SHIFT	I	S ╮
192	TAN				TAN
193	ATN	A	SHIFT	T	A │
194	PEEK	P	SHIFT	E	P ‾
195	LEN				LEN
196	STR$	ST	SHIFT	R	ST ▁
197	VAL	V	SHIFT	A	V ♠
198	ASC	A	SHIFT	S	A ♥
199	CHR$	C	SHIFT	H	C │
200	LEFT$	LE	SHIFT	F	LE ▁
201	RIGHT$	R	SHIFT	I	R ╮
202	MID$	M	SHIFT	I	M ╮

Come puoi vedere la tabella è stata stampata con caratteri ingranditi, usando il codice di controllo 14.

Dalla tabella precedente mancano le seguenti parole chiave:

TIME **TIME$** **STATUS** **π** **GET#**

le prime tre sono abbreviate abitualmente in TI, TI$ e ST.

Per vedere come esse sono rappresentate in memoria, abbiamo scritto un programma, nel quale sono usate queste parole chiave, e poi, il programma stesso, lista la sua rappresentazione in memoria.

```
1 REM ALTRITOKENS
2 GOTO100
10 PRINTTI
20 PRINTTI$
30 PRINTST
40 PRINTπ
50 GET#3,A$
60 STOP
100 A=PEEK(43)+256*PEEK(44)
105 OPEN4,4:CMD4
107 PRINT"LISTA BYTE PROGRAMMA"
110 PRINT:K=0
120 FORJ=1TO4:X=PEEK(A+K):PRINTX;
123 K=K+1:NEXTJ
125 PRINT:N=0:PRINT"     ";
126 X=PEEK(A+K):PRINTX;:N=N+1
127 IFX=0THEN130
128 IFN>8THENPRINT:N=0:PRINT"     ";
129 K=K+1:GOTO126
130 PRINT:IFPEEK(A+K+1)=0THEN160
150 K=K+1:GOTO120
160 PRINT#4:CLOSE4:STOP
```

Se osservi il listato vedi che il programma contiene un errore; infatti alla linea 50 si opera con una GET#, su un canale non aperto. Per questa ragione le istruzione dalla linea 10 alla linea 60 non vengono mai eseguite, ma sono state scritte solo per analizzare la loro memorizzazione. Alla linea 2, con GOTO 100, si va ad eseguire la routine che stampa le istruzioni byte dopo byte, come si trovano in memoria. La

routine da 100 a 160 può essere sfruttata come sottoprogramma sostituendo allo STOP finale un RETURN, ed essere usata in altri programmi. Essa sfrutta il fatto che il programma termina con un byte contenente zero, subito dopo lo zero finale dell'ultima istruzione. Il listato riporta su una riga i due byte che formano il LINK (puntano cioè all'indirizzo della istruzione seguente) e i due byte che danno il numero di linea Basic. Nella riga successiva è listata l'istruzione, andando a capo dopo 9 byte e alla fine della istruzione.
Andiamo a vedere la linea che inizia con: 36 8 10 0; essa è la linea di programma 10. Il 153 che segue è il codice di PRINT, 84 e 73 sono i codici ASCII delle lettere T e I. Nella istruzione seguente vediamo che per TI$ sono presenti i codici: 84, 73 e 36, corrispondenti ai tre caratteri della parola.
Nella istruzione seguente vediamo che sono presenti i codici di S e di T: 83 e 84. Proseguendo, troviamo per π il codice 255.
Infine per GET# troviamo 161, codice di GET, seguito da 35, codice di #.
Puoi proseguire nell'analisi di questo tabulato e soddisfare alcune curiosità, sempre che la cosa ti interessi.

RISULTATI PROGRAMMA PRECEDENTE

```
LISTA BYTE PROGRAMMA

 19  8  1  0
    143  32  65  76  84  82  73  84  79
    75  69  78  83  0
 28  8  2  0
    137  49  48  48  0
 36  8  10  0
    153  84  73  0
 45  8  20  0
    153  84  73  36  0
 53  8  30  0
    153  83  84  0
 60  8  40  0
    153  255  0
 71  8  50  0
    161  35  51  44  65  36  0
 77  8  60  0
    144  0
 99  8  100  0
    65  178  194  40  52  51  41  170  50
    53  54  172  194  40  52  52  41  0
 111  8  105  0
```

```
    159  52  44  52  58  157  52  0
139  8  107  0
    153  34  76  73  83  84  65  32  66
    89  84  69  32  80  82  79  71  82
    65  77  77  65  34  0
149  8  110  0
    153  58  75  178  48  0
173  8  120  0
    129  74  178  49  164  52  58  88  178
    194  40  65  170  75  41  58  153  88
    59  0
186  8  123  0
    75  178  75  170  49  58  130  74  0
204  8  125  0
    153  58  78  178  48  58  153  34  32
    32  32  34  59  0
227  8  126  0
    88  178  194  40  65  170  75  41  58
    153  88  59  58  78  178  78  170  49
    0
240  8  127  0
    139  88  178  48  167  49  51  48  0
7  9  128  0
    139  78  177  56  167  153  58  78  178
    48  58  153  34  32  32  32  34  59
    0
22  9  129  0
    75  178  75  170  49  58  137  49  50
    54  0
44  9  130  0
    153  58  139  194  40  65  170  75  170
    49  41  178  48  167  49  54  48  0
59  9  150  0
    75  178  75  170  49  58  137  49  50
    48  0
71  9  160  0
    152  52  58  160  52  58  144  0
```

E' evidente che le routine del sistema quando incontrano la parola chiave GET, vanno ad analizzare il carattere successivo per stabilire se è un #; inoltre ci devono essere dei controlli per stabilire se sono in presenza delle altre quattro parole chiave sopra citate.

CAPITOLO 8

UTILIZZO DELLA MEMORIA E CONFIGURAZIONI

8.1 PREMESSA

Abbiamo già accennato nel Capitolo 1 alla tecnica di gestione della memoria mediante puntatori situati nelle prime pagine della memoria RAM.

Ora ci occupiamo più dettagliatamente dei puntatori che gestiscono la parte di memoria RAM occupata dal programma in Basic e dalle sue variabili. Segue un programma che stampa sul video e, se si vuole, anche sulla stampante, il contenuto di 7 puntatori. Nel programma sono caricati con delle frasi DATA sia le stringhe di descrizione dei puntatori, che gli indirizzi dei relativi byte. Al listato del programma fa seguito il risultato su stampante.

```
1 REM PUNTBASIC
2 INPUT"RISULTATI SU STAMPANTE? (S/N) ";R$
3 PRINT:IFR$="N"THEN8
4 OPEN4,4
5 PRINT#4,"***CONTENUTO PUNTATORI A 2 BYTE***"
6 PRINT#4
8 D$="":READD$,B,A
9 IFD$="*"THEN40
10 Z=256*PEEK(A)+PEEK(B)
20 PRINTD$;SPC(2);"(";B;"/";A;")";Z
25 IFR$="N"THEN35
30 PRINT#4,D$;SPC(2);"(";B;"/";A;")";Z
31 PRINT#4
35 GOTO8
40 IFR$="S"THENCLOSE4
45 STOP
100 DATA"INIZIO PROGRAMMA BASIC",43,44
101 DATA"INIZIO VARIABILI BASIC",45,46
102 DATA"INIZIO ARRAY BASIC",47,48
103 DATA"FINE ARRAY BASIC + 1",49,50
```

```
104 DATA"IND. FONDO AREA STRINGHE",51,52
105 DATA"PUNTATORE STRINGHE",53,54
106 DATA"INDIRIZZO PIU' ALTO BASIC",55,56
107 DATA"*",0,0
```

RISULTATI PROGRAMMA

```
***CONTENUTO PUNTATORI A 2 BYTE***

INIZIO PROGRAMMA BASIC  ( 43 / 44 ) 2049

INIZIO VARIABILI BASIC  ( 45 / 46 ) 2611

INIZIO ARRAY BASIC  ( 47 / 48 ) 2646

FINE ARRAY BASIC + 1  ( 49 / 50 ) 2646

IND. FONDO AREA STRINGHE  ( 51 / 52 ) 40959

PUNTATORE STRINGHE  ( 53 / 54 ) 40960

INDIRIZZO PIU' ALTO BASIC  ( 55 / 56 ) 40960
```

I risultati che vedi si riferiscono alla situazione della memoria quando è presente questo programma. Esso non definisce variabili con indice (chiamate ARRAY) e quindi i relativi puntatori di inizio e fine sono allo stesso valore, puntano subito dopo la fine delle variabili.
Come vedi il più alto indirizzo del Basic è 40960 e il programma inizia in 2049. La zona dedicata al corpo delle stringhe comincia all'indirizzo 40959 e viene usata per indirizzi decrescenti. Le stringhe hanno le loro testate di definizione tra le altre variabili; nella prima zona se sono variabili stringa singole, nella seconda se sono variabili stringa con indice.

Segue un altro listato, relativo a un programma ottenuto modificando il precedente con l'aggiunta di un sottoprogramma in 200, che opera un dimensionamento e definisce due nuove stringhe; e con la modifica della linea 3, dove è stata aggiunta la chiamata al sottoprogramma. Al listato seguono i risultati su stampante.

```
1 REM PUNTBASMOD
2 INPUT"[illegible]RISULTATI SU STAMPANTE? (S/N) ";R$
3 PRINT:GOSUB200:IFR$="N"THEN8
```

```
4 OPEN4,4
5 PRINT#4,"***CONTENUTO PUNTATORI A 2 BYTE***"
6 PRINT#4
8 D$="":READD$,B,A
9 IFD$="*"THEN40
10 Z=256*PEEK(A)+PEEK(B)
20 PRINTD$;SPC(2);"(";B;"/";A;")";Z
25 IFR$="N"THEN35
30 PRINT#4,D$;SPC(2);"(";B;"/";A;")";Z
31 PRINT#4
35 GOTO8
40 IFR$="S"THENCLOSE4
45 STOP
100 DATA"INIZIO PROGRAMMA BASIC",43,44
101 DATA"INIZIO VARIABILI BASIC",45,46
102 DATA"INIZIO ARRAY BASIC",47,48
103 DATA"FINE ARRAY BASIC + 1",49,50
104 DATA"IND. FONDO AREA STRINGHE",51,52
105 DATA"PUNTATORE STRINGHE",53,54
106 DATA"INDIRIZZO PIU' ALTO BASIC",55,56
107 DATA"*",0,0
200 DIMN(20):M$="UNA STRINGA"
201 N$="UN'ALTRA STRINGA"
202 RETURN
```

RISULTATI PROGRAMMA VARINMEM

```
***CONTENUTO PUNTATORI A 2 BYTE***

INIZIO PROGRAMMA BASIC  ( 43 / 44 ) 2049

INIZIO VARIABILI BASIC  ( 45 / 46 ) 2677

INIZIO ARRAY BASIC  ( 47 / 48 ) 2726

FINE ARRAY BASIC + 1  ( 49 / 50 ) 2838

IND. FONDO AREA STRINGHE  ( 51 / 52 ) 40959

PUNTATORE STRINGHE  ( 53 / 54 ) 40960

INDIRIZZO PIU' ALTO BASIC  ( 55 / 56 ) 40960
```

Come puoi vedere, avendo allungato il programma, l'inizio delle variabili si è spostato. Inoltre, avendo definito degli array i due puntatori relativi hanno valori diversi.

Se esegui in immediato il comando SYS 64738, cioè ripristini le condizioni iniziali del calcolatore, e, sempre in immediato vai a leggere con il comando PEEK il valore dei puntatori:

PRINT PEEK(k)+256*PEEK(k+1)

ponendo per k e k+1 i valori visti troverai che:

43/44	contengono	2049
45/46	"	2051
47/48	"	2051
49/50	"	2051
51/52	"	40960
53/54	"	o
55/56	"	40960

La figura 8.1 esemplifica l'uso della memoria da parte di un programma.

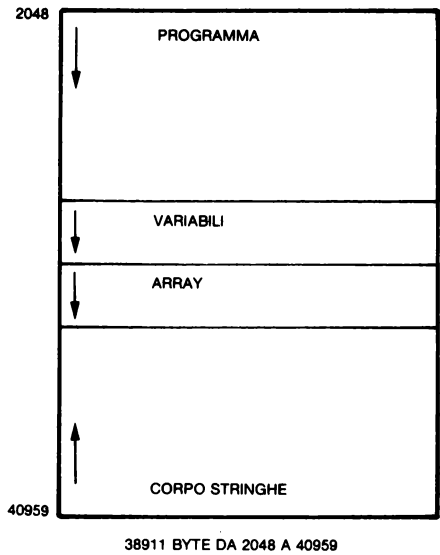



Figura 8.1 Utilizzo memoria RAM programma

Riportiamo un esempio abbastanza interessante di uso avanzato del linguaggio Basic.

I puntatori contenuti nei byte 63/64 e 65/66 controllano la gestione dei dati immagazzinati con le frasi DATA; il primo contiene il numero della linea DATA in uso e il secondo contiene l'indirizzo di memoria dell'elemento disponibile. Il sistema consente con l'istruzione RESTORE di riposizionarsi all'inizio dei dati. Se si desidera imparare a riposizionarsi in un punto desiderato della lista dei dati si può considerare il programma che segue, nel quale si leggono con delle istruzioni PEEK i contenuti dei puntatori in determinati momenti e più avanti si ripristinano i valori letti e memorizzati per riportarsi nella situazione precedente.

```
1 REM RESTORE
10 DATA0,1,2,3,4,5,6,7,8,9
11 DATA10,11,12,13,14,15,16,17,18,19
12 DATA20,21,22,23,24,25,26,27,28,29
15 OPEN4,4:CMD4
20 PRINT"VALORI INIZIALI"
25 GOSUB100
27 PRINT"LETTURA 10 NUMERI"
30 FORK=0TO9:READW:PRINTW;:NEXTK:PRINT
35 PRINT"DOPO 10 READ":GOSUB100:GOSUB150
37 PRINT"LETTURA ALTRI 5 NUMERI"
40 FORK=0TO4:READW:PRINTW;:NEXTK:PRINT
45 PRINT"DOPO ALTRI 5 READ":GOSUB100:GOSUB160
50 PRINT"DOPO RIPOSIZIONAMENTO  INIZIO LINEA 11"
53 GOSUB100:GOSUB150
54 PRINT"LETTURA 5 NUMERI"
55 FORK=0TO4:READW:PRINTW;:NEXTK:PRINT:GOSUB150
57 PRINT"LETTURA ULTIMI"
59 FORK=0TO7:READW:PRINTW;:NEXTK:PRINT
60 FORK=0TO6:READW:PRINTW;:NEXTK:PRINT
61 PRINT"VALORI FINALI PUNTATORI":GOSUB100
63 PRINT"RIPOSIZIONAMENTO META' LINEA 11"
64 GOSUB160:GOSUB100
65 PRINT"LETTURA 5 NUMERI"
70 FORK=0TO4:READW:PRINTW;:NEXTK:PRINT
99 PRINT#4:CLOSE4:STOP
100 X=PEEK(63)+256*PEEK(64)
101 Y=PEEK(65)+256*PEEK(66)
102 PRINT"PUNTATORE 63/64:  ";X
103 PRINT"PUNTATORE 65/66:  ";Y:RETURN
150 Z1%=PEEK(63):Z2%=PEEK(64)
151 Z3%=PEEK(65):Z4%=PEEK(66):RETURN
160 POKE63,Z1%:POKE64,Z2%
161 POKE65,Z3%:POKE66,Z4%:RETURN
```

RISULTATI PROGRAMMA RESTORE

```
VALORI INIZIALI
PUNTATORE 63/64:  0
PUNTATORE 65/66:  2048
LETTURA 10 NUMERI
 0  1  2  3  4  5  6  7  8  9
DOPO 10 READ
PUNTATORE 63/64:  10
PUNTATORE 65/66:  2087
LETTURA ALTRI 5 NUMERI
 10  11  12  13  14
DOPO ALTRI 5 READ
PUNTATORE 63/64:  11
PUNTATORE 65/66:  2107
DOPO RIPOSIZIONAMENTO  INIZIO LINEA 11
PUNTATORE 63/64:  10
PUNTATORE 65/66:  2087
LETTURA 5 NUMERI
 10  11  12  13  14
LETTURA ULTIMI
 15  16  17  18  19  20  21  22
 23  24  25  26  27  28  29
VALORI FINALI PUNTATORI
PUNTATORE 63/64:  12
PUNTATORE 65/66:  2157
RIPOSIZIONAMENTO META' LINEA 11
PUNTATORE 63/64:  11
PUNTATORE 65/66:  2107
LETTURA 5 NUMERI
 15  16  17  18  19
```

Il programma è stato caricato e lanciato subito dopo l'accensione del calcolatore; se lo fai girare più volte partendo da una situazione diversa potrai trovare una differenza nel contenuto iniziale del puntatore in 63/64; infatti esso non viene azzerato per effetto del comando RUN.
Il sottoprogramma in 150 memorizza i 4 byte in Z1%, Z2%, Z3% e Z4%; quello in 160 ripristina i valori dei puntatori prelevandoli dalle 4 variabili citate.
Il sottoprogramma in 100 calcola i valori dei puntatori e li stampa.

8.2 VARIABILI IN MEMORIA

L'interprete Basic assegna lo spazio di memoria alle variabili al momento della loro definizione.

La definizione delle variabili singole si ha quando esse sono citate la prima volta nel programma in una delle seguenti condizioni:
.. compaiono a sinistra di un uguale;
.. sono listate dopo la parola chiave INPUT;
.. sono listate dopo la parola chiave READ.

La definizione delle variabili con indice si ha:
.. quando si definiscono con la frase DIM;
.. quando viene citato la prima volta uno degli elementi e si ha un dimensionamento implicito a 10.
Il dimensionamento produce l'assegnazione del valore zero alle variabili numeriche, e del valore stringa-nulla alle variabili stringa.

Ricordiamo che il nome di una variabile può essere al massimo di due caratteri, di cui il primo deve essere una lettera e il secondo può essere una lettera o una cifra. Per ogni variabile singola il sistema usa 7 byte, i primi 2 servono per il nome che è considerato sempre di 2 caratteri, gli altri 5 sono usati in modo diverso a seconda del tipo della variabile. Nella descrizione delle variabili chiamiamo byte 1 il primo a sinistra e byte 7 l'ultimo a destra. Inoltre con HI intendiamo riferirci al valore del byte più significativo e con LO al valore del byte meno significativo.

VARIABILE INTERA: nome di al massimo 2 caratteri seguiti dal suffisso %.
.. byte 1: codice ASCII primo carattere nome + 128;
.. byte 2: codice ASCII secondo carattere + 128, se il nome è di un solo carattere, compare solo 128;
.. byte 3: byte HI del numero intero;
.. byte 4: byte LO del numero intero;
.. byte 5: non usato contiene 0;
.. byte 6: non usato contiene 0;
.. byte 7: non usato contiene 0.

VARIABILE REALE: nome di al massimo 2 caratteri senza suffisso.
.. byte 1: codice ASCII del primo caratere del nome;
.. byte 2: codice ASCII del secondo carattere del nome, o 0 se il nome è di un solo carattere;
.. byte 3: esponente del numero floating point;
.. byte 4: byte HI della mantissa del numero;

.. byte 5: byte successivo della mantissa;
.. byte 6: byte successivo della mantissa;
.. byte 7: byte LO della mantissa.

VARIABILE STRINGA: nome di al massimo 2 caratteri con il suffisso $.
.. byte 1: codice ASCII del primo carattere del nome;
.. byte 2: codice ASCII del secondo carattere del nome + 128, o solo 128 se il nome è di un solo carattere;
.. byte 3: contatore dei caratteri che compongono la stringa, massimo 255;
.. byte 4: byte LO del puntatore al corpo della stringa;
.. byte 5: byte HI del puntatore al corpo della stringa;
.. byte 6: non usato contiene 0;
.. byte 7: non usato contiene 0.
Nel momento in cui viene definita la prima volta la variabile stringa vengono assegnati questi 7 byte e il puntatore al corpo della stringa ha il valore che è possibile in quel momento, in dipendenza dalla situazione della zona di memoria assegnata ai corpi delle stringhe. Quando la stessa variabile stringa riceve un nuovo contenuto, in questi byte di testata vengono modificati: il byte 3, contenente il numero di caratteri e che quindi può variare, e i byte 4 e 5 che definiscono il puntatore a una nuova posizione di memoria. La memoria viene assegnata dinamicamente ai corpi delle stringhe durante l'uso del programma. Quando, dopo molti cambiamenti, la zona di memoria sembra esaurita, il sistema fa una operazione di compattamento, cioè sposta tutti i corpi delle stringhe verso il fondo della memoria, infatti si creano continuamente dei buchi, via via che si hanno le nuove assegnazioni. In tale modo il sistema si procura di nuovo memoria per i corpi delle stringhe; l'operazione viene chiamata "garbage collection". Può tuttavia succedere che durante l'uso di un programma si esaurisca la memoria disponibile per i corpi delle stringhe; tutte le stringhe vengono scritte alla massima lunghezza. L'unico rimedio in tale situazione è quello di porre dei limiti alle lunghezze delle stringhe, tagliandole quando vengono ricevute.
Durante l'esecuzione di un programma puoi avvertire delle soste inconsuete; esse dipendono dal fatto che il calcolatore ha esaurito la zona di memoria per i corpi delle stringhe e opera il compattamento.

Per le VARIABILI CON INDICE vale la stessa regola di quelle singole per la formazione dei nom non possono esserci confusioni dato che sono memorizzate in una zona a parte. Il numero dei byte occupati dipende dal numero delle dimensioni e dal valore di ogni dimensione. Si ha sempre una testata di 5 byte, seguita da una coppia di byte per ogni dimensione e da:
.. 2 byte per ogni elemento per le variabili intere;
.. 5 byte per ogni elemento per le variabili decimali (floating-point);
.. 3 byte per ogni elemento per le variabili stringa, più, in altra zona di memoria, un byte per ogni carattere di ogni elemento.

ARRAY DI VARIABILI PER NUMERI INTERI: nome di al massimo due caratteri seguito dal suffisso %.
.. byte 1: codice ASCII del primo carattere del nome + 128;
.. byte 2: codice ASCII del secondo carattere del nome + 128 o solo 128;
.. byte 3: byte LO del contatore del numero totale dei byte occupati;
.. byte 4: byte HI del contatore del numero totale dei byte occupati;
.. byte 5: numero delle dimensioni;
.. una coppia di byte come contatore (HI-LO) del numero degli elementi dell'ultima dimensione (valore massimo dell'indice +1);
.. una coppia di byte come contatore del numero degli elementi della penultima dimensione;
...
...
.. una coppia di byte come contatore del numero degli elementi della prima dimensione;
.. una coppia di byte (HI-LO) per ogni elemento.

Come vedi usando un array per definire le variabili intere si ha un risparmio di memoria rispetto alla definizione dello stesso numero di variabili come variabili singole. Supponiamo di definire 10 variabili singole intere e un array di 10 elementi interi. Nel primo caso occupiamo 7x10=70 byte; nel secondo 5+2x10+2x10=45 byte. Su un numero elevato di variabili questo può essere conveniente, anche se nel computo si deve tenere presente che, ogni volta che si richiama una variabile intera nel programma, si usano 2 o 3 byte per citare il nome della singola, mentre per quella con indice, altre ai 2 o 3 byte del nome ne servono 2 per aprire e chiudere le parentesi e da 1 a, diciamo, 4 per l'indice (pensiamo a un array con un solo indice).

ARRAY DI VARIABILI PER NUMERI FLOATING-POINT: nome di al massimo due caratteri senza suffisso.
Vale quanto detto per le variabili intere, salvo per il nome che è espresso come abbiamo visto per quelle floating-point singole, e per il numero di byte occupato per ogni elemento che sale a 5.
In questo caso non si ha risparmio di memoria rispetto alla definizione delle variabili singole; si deve ricorrere a questo tipo di array quando è utile per lo svolgimento della programmazione.

ARRAY DI VARIABILI PER STRINGHE: nome di al massimo due caratteri seguito dal suffisso $.
Vale quanto detto per le variabili intere, salvo per il nome che è espresso come abbiamo visto per le variabili stringa singole, e per il numero di byte occupato per ogni elemento che sale a 3. Questi 3 byte sono così utilizzati: il primo come contatore del numero di caratteri dell'elemento, che può arrivare al massimo a 255; gli altri due byte (LO-HI) del puntatore al CORPO della stringa che si trova nella zona di memoria dedicata allo scopo.

Per vedere quanto abbiamo detto in memoria, abbiamo preparato un programma che crea variabili di ogni tipo e, servendosi dei puntatori, va a leggere e stampare i contenuti delle diverse zone della memoria.

```
1 REM VARINMEM
3 M=0:N=0:Y=0:Z=0
5 DIMD(6),E%(5,4),F$(5,3,2)
7 A=0:FORK=1TO50 A=A+1:NEXTK
9 FORK=0TO6:D(K)=K*3:NEXTK
11 B%=0:FORI=1TO50STEP2:B%=B%+2:NEXTI
13 FORK=0TO5:FORI=0TO4
15 E%(K,I)=K*1+9:NEXTI,K
17 C$="FINE"
19 FORK=0TO5:FORJ=0TO3:FORI=0TO2
21 READY$:F$(K,J,I)=C$+Y$
23 NEXTI,J,K
25 Y=PEEK(45)+256*PEEK(46)
27 OPEN4,4:CMD4
29 PRINT:PRINT"VARIABILI":M=Y
31 PRINTM;"**"
33 FORI=0TO10:FORN=0TO6
35 Y=PEEK(45)+256*PEEK(46)
37 PRINTPEEK(M+N);"  ";:NEXTN:PRINT:PRINT
39 M=M+7:NEXTI
41 Y=PEEK(47)+256*PEEK(48)
43 Z=PEEK(49)+256*PEEK(50)
45 PRINT:PRINT"ARRAY":PRINTY;"**":K=Y
51 M=PEEK(K+2)+256*PEEK(K+3)
57 I=0:FORJ=KTOK+M-1
59 PRINTPEEK(J);" ";:I=I+1
61 IFI=7THENPRINT:I=0
63 NEXTJ:K=K+M:IFK=ZTHEN67
65 PRINT:GOTO51
67 PRINT:PRINT"FINE ARRAY: ";PEEK(Z)
79 PRINT:PRINT:PRINT"CORPO STRINGHE":PRINT
81 Y=PEEK(51)+256*PEEK(52)
82 Z=PEEK(55)+256*PEEK(56)
83 I=0:FORK=YTOY+50:PRINTK;"*";PEEK(K);
84 I=I+1:IFI=3THENPRINT:I=0
85 NEXTK
86 I=0:FORK=Z-50TOZ:PRINTK;"*";PEEK(K);
87 I=I+1:IFI=3THENPRINT:I=0
88 NEXTK
```

```
100 PRINT#4:CLOSE4:STOP
110 DATAAAA,BBB,CCC,DDD,EEE,FFF,GGG,III,LLL
111 DATAA,B,C,D,E,F,G,H,I,J,K,L,M,N,O,P,Q,R,S,T
112 DATAU,V,W,X,Y,Z,A1,B1,C1,D1,E1,F1,G1,H1,I1
113 DATAJ1,K1,L1,M1,N1,O1,P1,Q1,R1,S1,T1,U1,V1
114 DATAW1,X1,Y1,Z1,ABC,DEF,GHI,JKL,MNO,PQR,STU
115 DATAVWX,YZZ,PIPPO,PLUTO
```

RISULTATI PROGRAMMA VARINMEM

```
VARIABILI
 3109 **
 77    0    140    66    80    0    0

 78    0    130    64    0    0    0

 89    0    140    66    80    0    0

 90    0    0    0    0    0    0

 65    0    134    72    0    0    0

 75    0    131    64    0    0    0

 194    128    0    50    0    0    0

 73    0    131    96    0    0    0

 67    128    4    191    8    0    0

 74    0    131    0    0    0    0

 89    128    5    29    12    0    0


ARRAY
 3186 **
 68   0   42   0   1   0   7
 0   64   0   0   0   130   64
 0   0   0   131   64   0   0
 0   132   16   0   0   0   132
 64   0   0   0   132   112   0
 0   0   133   16   0   0   0
```

```
197   128   69   0   2   0   5
0   6   0   9   0   10   0
11   0   12   0   13   0   14
0   9   0   10   0   11   0
12   0   13   0   14   0   9
0   10   0   11   0   12   0
13   0   14   0   9   0   10
0   11   0   12   0   13   0
14   0   9   0   10   0   11
0   12   0   13   0   14
70   128   227   0   3   0   3
0   4   0   6   7   249   159
5   173   159   5   113   159   6
51   159   6   235   158   6   163
158   7   228   159   5   158   159
5   98   159   6   33   159   6
217   158   7   142   158   7   207
159   5   143   159   5   83   159
6   15   159   6   199   158   7
121   158   5   188   159   5   128
159   5   68   159   6   253   158
6   181   158   7   100   158   7
242   159   5   168   159   5   108
159   6   45   159   6   229   158
7   156   158   7   221   159   5
153   159   5   93   159   6   27
159   6   211   158   7   135   158
7   200   159   5   138   159   5
78   159   6   9   159   6   193
158   7   114   158   5   183   159
5   123   159   5   63   159   6
247   158   6   175   158   9   91
158   7   235   159   5   163   159
5   103   159   6   39   159   6
223   158   7   149   158   7   214
159   5   148   159   5   88   159
6   21   159   6   205   158   7
128   158   7   193   159   5   133
159   5   73   159   6   3   159
6   187   158   7   107   158   5
178   159   5   118   159   6   57
159   6   241   158   6   169   158
9   82   158
FINE ARRAY:   0
```

```
CORPO STRINGHE

 40530 * 70  40531 * 73  40532 * 78
 40533 * 69  40534 * 80  40535 * 76
 40536 * 85  40537 * 84  40538 * 79
 40539 * 70  40540 * 73  40541 * 78
 40542 * 69  40543 * 80  40544 * 73
 40545 * 80  40546 * 80  40547 * 79
 40548 * 70  40549 * 73  40550 * 78
 40551 * 69  40552 * 89  40553 * 90
 40554 * 90  40555 * 70  40556 * 73
 40557 * 78  40558 * 69  40559 * 86
 40560 * 87  40561 * 88  40562 * 70
 40563 * 73  40564 * 78  40565 * 69
 40566 * 83  40567 * 84  40568 * 85
 40569 * 70  40570 * 73  40571 * 78
 40572 * 69  40573 * 80  40574 * 81
 40575 * 82  40576 * 70  40577 * 73
 40578 * 78  40579 * 69  40580 * 77
 40910 * 73  40911 * 70  40912 * 73
 40913 * 78  40914 * 69  40915 * 71
 40916 * 71  40917 * 71  40918 * 70
 40919 * 73  40920 * 78  40921 * 69
 40922 * 70  40923 * 70  40924 * 70
 40925 * 70  40926 * 73  40927 * 78
 40928 * 69  40929 * 69  40930 * 69
 40931 * 69  40932 * 70  40933 * 73
 40934 * 78  40935 * 69  40936 * 68
 40937 * 68  40938 * 68  40939 * 70
 40940 * 73  40941 * 78  40942 * 69
 40943 * 67  40944 * 67  40945 * 67
 40946 * 70  40947 * 73  40948 * 78
 40949 * 69  40950 * 66  40951 * 66
 40952 * 66  40953 * 70  40954 * 73
 40955 * 78  40956 * 69  40957 * 65
 40958 * 65  40959 * 65  40960 * 148
```

Il programma definisce 11 variabili singole; esse vengono assegnate nel seguente ordine: M, N, Y, Z, A, K, B%, I, C$, J, Y$. Inoltre esso definisce 3 array, nel seguente ordine: D, E%, F$.
Per poter lavorare abbiamo dovuto definire all'inizio del programma, prima di andare a leggere il valore dei puntatori tutte le variabili, infatti (se osservi la Figura

8.1, te ne rendi subito conto), se si aggiunge una variabile singola, tutti gli array devono essere spostati in giù; questa è anche una perdita di tempo. Osserva la linea 3; in essa vengono definite 4 variabili singole, assegnando ad esse il valore zero. E' consigliabile fare all'inizio del programma sempre tutte le assegnazioni delle variabili singole, al limite ponendole a zero, poi passare a definire gli array. Operando in tale modo le variabili restano posizionate in memoria allo stesso posto per tutta la durata del programma; variano in modo dinamico solo le posizioni dei corpi delle stringhe.
Nel programma dalla linea 3 alla 23 si ha la creazione di tutte le variabili, anche di quelle che vengono usate come contatori per controllare i cicli FOR.
Per assegnare valori alla matrice di stringhe che ha 72 elementi (6x4x3=72), ci siamo serviti di frasi DATA dalla linea 110 alla 115, e del ciclo con READ dalla linea 19 alla linea 23. Nota la chiusura dei tre cicli FOR concatenati con una sola parola chiave NEXT seguita dai nomi delle 3 variabili di controllo.
Dalla linea 25 alla linea 39 si stampano i contenuti dei byte usati per le variabili singole. Si ricava dai byte 45 e 46 il valore del puntatore (3186) e si stampa seguito da due asterischi. Poi vengono stampati su ogni riga i 7 byte di ognuna delle 11 variabili singole. Puoi controllare quanto abbiamo detto all'inizio di questo paragrafo. Osserva le due linee che si riferiscono a stringhe, la terzultima e l'ultima, nella parte dei risultati intitolata VARIABILI. Vedi per C$ il nome: 67, 128, poi il 4 del terzo byte dice che la stringa ha 4 caratteri (FINE), i due byte successivi sono rispettivamente i byte LO e HI del puntatore al corpo della stringa (8x256+191=2239) che si trova incorporata nel programma alla linea 17. Per Y$ invece, che risulta essere di 5 caratteri, infatti l'ultima stringa trattata con READ è "PLUTO", si trova come valore del puntatore 12x256+29=3101, questo è l'indirizzo di memoria interno al programma dove inizia la parola "PLUTO" alla fine della linea 115. Dopo si ha lo zero di fine istruzione al byte 3106, i due byte 3107 e 3108 contengono i due zeri di fine programma e in 3109 iniziano le variabili.
Dalla linea 41 alla linea 67 si stampano gli array. Si calcola l'inizio degli array per mezzo del puntatore nei byte 47 e 48 e la fine più uno per mezzo del puntatore nei byte 49 e 50. Alla linea 51 si calcola M usando il terzo e il quarto byte; esso è il numero dei caratteri occupati dall'array in questa zona. Per il primo array D(6) sono occupati in tutto 42 byte. Il quinto byte dice che si ha una dimensione, il sesto e il settimo byte (prima HI e poi LO) dicono che l'array ha 7 elementi. Seguono poi 5 byte per ogni elemento; se non riesci a ritrovare i numeri che devono essere: 0, 3, 6, 9, 12, 15, 18, vai a rivedere nel Capitolo 7 la parte dove vengono svolti i calcoli relativi alla mantissa dei numeri floating-point.
Per E%(5,4) trovi dopo il nome, che occupa in tutto 69 byte, che ha 2 dimensioni, che l'ultima dimensione ha 5 elementi, che la prima dimensione ha 6 elementi; dopo trovi gli elementi che occupano 2 byte ciascuno.
Per F$(5,3,2), che occupa in tutto 227 byte, fissiamo la nostra attenzione su qualche elemento. Il primo (byte 12,13 e 14) ha 7 caratteri e il corpo della stringa si trova in 159x256+249=40953; lo puoi vedere in fondo ai risultati. Partendo dal byte 40953,

troviamo i codici: 70, 73, 78, 69, 65, 65, 65, che corrispondono a "FINEAAA". Ti rendi conto del modo come vengono sistemate le stringhe in memoria; questa è stata assegnata per prima e sta sul fondo della memoria. Consideriamo l'ultimo elemento (9, 82, 158), esso ha 9 caratteri e la stringa si trova in 158x256+82=40530; puoi vederla all'inizio della parte dei risultati intitolata CORPO STRINGHE. Partendo dal byte 40530 troviamo: 70, 73, 78, 69, 80, 76, 85, 84, 79, che corrisponde a "FINEPLUTO", ultima stringa assegnata.
Nelle linee da 79 a 88 si stampa una parte del corpo delle stringhe e precisamente gli ultimi 51 byte e i primi 51 byte.
Osserva che tra le variabili singole si trovano anche le variabili di controllo dei cicli FOR; esse si presentano come variabili decimali e non si differenziano dalle altre variabili dello stesso tipo. Quando viene eseguito un ciclo FOR le informazioni necessarie per gestirlo vengono memorizzate nell'area STACK del sistema, mentre le variabili di controllo restano in zona variabili.

Nella definizione delle variabili singole si ha spreco di memoria per gli interi e per le stringhe. Infatti per gli interi restano inutilizzati gli ultimi 3 byte dei 7 occupati e per le stringhe gli ultimi 2. Abbiamo preparato il programma UTILIZZO per farti vedere come puoi andare a scrivere qualcosa nei byte inutilizzati senza disturbare le variabili. Naturalmente devi procurarti gli indirizzi dei byte e scriverci dentro usando l'istruzione POKE.

```
1 REM UTILIZZO
10 REM DEFINIZIONE 2 VAR. INTERE
20 A%=5:B%=7
25 REM DEFINIZIONE 2 VAR. STRINGA
30 A$="PROVA":B$="RISPARMIO"
35 X=PEEK(45)+256*PEEK(46)
40 Y%=A%:GOSUB100:Y=X+4:GOSUB101
45 Y%=B%:GOSUB100:Y=X+11:GOSUB101
50 Y%=ASC(A$):POKEX+19,Y%
55 Y%=ASC(B$):POKEX+26,Y%
56 OPEN4,4:CMD4
60 PRINT"A%=";A%,"B%=";B%
65 PRINT"A$="A$,"B$="B$
67 PRINT"VARIABILI IN MEMORIA"
69 FORZ=0TO21STEP7
70 FORY=0TO6:PRINTPEEK(Z+Y+X);
71 NEXTY:PRINT:NEXTZ
75 PRINT#4:CLOSE4:STOP
100 Y%=Y%↑2:RETURN
101 POKEY,INT(Y%/256)
102 POKE(Y+1),Y%-PEEK(Y):RETURN
```

RISULTATI PROGRAMMA UTILIZZO

```
A%= 5              B%= 7
A$=PROVA              B$=RISPARMIO
VARIABILI IN MEMORIA
 193  128  0  5  0  25  0
 194  128  0  7  0  49  0
 65  128  5  103  8  80  0
 66  128  9  114  8  82  0
```

Il programma definisce per prime le 4 variabili che interessano; esse sono A%, B%, A$ e B$. In tale modo le variabili occupano quattro gruppi di 7 byte subito all'inizio dell'area variabili. Il sottoprogramma in 100 calcola il quadrato del numero che sta in Y% e lo pone ancora in Y%. Il sottoprogramma in 101 va a scrivere Y% nei 2 byte di indirizzo Y e Y+1.
Le variabili vengono stampate nel modo normale anche dopo aver usato i byte di coda e, come vedi, non sono disturbate. Dopo vengono mostrati i contenuti della memoria e puoi vedere i quadrati dei due numeri: 25 e 49. Per le due stringhe invece vedi il codice ASCII del primo carattere scritto nel penultimo byte.
Questo è un uso abbastanza sofisticato della memoria del calcolatore e abbiamo preparato l'esempio per far comprendere come lavora il sistema.

8.3 PROGRAMMA IN MEMORIA

Il programma Basic inizia al byte 2049, puntato dai due byte 43 e 44 della pagina zero.
La struttura delle istruzioni è la seguente:

.. 2 byte di LINK (legamento), che contengono l'indirizzo della linea di programma successiva; l'ultima istruzione del programma ha i due byte di link a zero binario (tutti bit zero); il primo byte è il LO, il secondo lo HI;
.. 2 byte contenenti il numero della linea Basic; il primo LO e il secondo HI;
.. seguono i byte che contengono la linea Basic, nella quale le parole chiave sono compresse e vengono espresse dal codice in un byte, mentre tutte le altre parti componenti la linea sono espresse carattere per carattere;
.. la linea termina con un byte a zero binario.

Da quanto detto risulta che tre zeri binari vicini segnalano la fine del programma. Se la linea Basic contiene degli spazi, essi vengono mantenuti con spreco di memoria. Non viene conservato lo spazio tra il numero di linea e il primo carattere della frase; questo spazio viene aggiunto in fase di LIST del programma.

Segue un breve programma che stampa se stesso, a scopo dimostrativo.

```
1 REM PRGINMEM
5 OPEN4,4:CMD4:PRINT"PROGRAMMA IN MEMORIA":PRINT
10 X=256*PEEK(44)+PEEK(43)
15 Y1=PEEK(X):Y2=PEEK(X+1)
17 IFY1+Y2=0THENPRINTX;"**";Y1;Y2:GOTO60
20 PRINTX;"**";Y1;Y2;"LINK=";Y2*256+Y1
25 Y1=PEEK(X+2):Y2=PEEK(X+3)
30 PRINT"NUMERO LINEA: ";Y2*256+Y1
35 X=X+4:C=0
37 Y1=PEEK(X):IFY1=0THEN55
40 PRINTY1;"    ";:X=X+1:C=C+1
45 IFC=6THENPRINT:C=0
50 GOTO37
55 PRINTY1:X=X+1:GOTO15
60 PRINT#4:CLOSE4:STOP
```

RISULTATI PROGRAMMA PRGINMEM

```
PROGRAMMA IN MEMORIA

 2049 ** 16  8 LINK= 2064
NUMERO LINEA:  1
 143      32      80      82      71      73
 78      77      69      77      0
 2064 ** 54  8 LINK= 2102
NUMERO LINEA:  5
 159      52      44      52      58      157
 52      58      153      34      80      82
 79      71      82      65      77      77
 65      32      73      78      32      77
 69      77      79      82      73      65
 34      58      153      0
 2102 ** 76  8 LINK= 2124
NUMERO LINEA:  10
 88      178      50      53      54      172
 194      40      52      52      41      170
 194      40      52      51      41      0
 2124 ** 98  8 LINK= 2146
```

```
NUMERO LINEA:  15
 89      49      178     194     40      88
 41      58      89      50      178     194
 40      88      170     49      41      0
 2146 ** 129  8 LINK= 2177
NUMERO LINEA:  17
 139     89      49      170     89      50
 178     48      167     153     88      59
 34      42      42      34      59      89
 49      59      89      50      58      137
 54      48      0
 2177 ** 165  8 LINK= 2213
NUMERO LINEA:  20
 153     88      59      34      42      42
 34      59      89      49      59      89
 50      59      34      76      73      78
 75      61      34      59      89      50
 172     50      53      54      170     89
 49      0
 2213 ** 189  8 LINK= 2237
NUMERO LINEA:  25
 89      49      178     194     40      88
 170     50      41      58      89      50
 178     194     40      88      170     51
 41      0
 2237 ** 221  8 LINK= 2269
NUMERO LINEA:  30
 153     34      78      85      77      69
 82      79      32      76      73      78
 69      65      58      32      34      59
 89      50      172     50      53      54
 170     89      49      0
 2269 ** 235  8 LINK= 2283
NUMERO LINEA:  35
 88      178     88      170     52      58
 67      178     48      0
 2283 ** 0  9 LINK= 2304
NUMERO LINEA:  37
 89      49      178     194     40      88
 41      58      139     89      49      178
 48      167     53      53      0
 2304 ** 27  9 LINK= 2331
```

```
NUMERO LINEA:  40
 153      89      49      59      34      32
 32       32      34      59      58      88
 178      88      170     49      58      67
 178      67      170     49      0
 2331 ** 42  9 LINK= 2346
NUMERO LINEA:  45
 139      67      178     54      167     153
 58       67      178     48      0
 2346 ** 50  9 LINK= 2354
NUMERO LINEA:  50
 137      51      55      0
 2354 ** 68  9 LINK= 2372
NUMERO LINEA:  55
 153      89      49      58      88      178
 88       170     49      58      137     49
 53       0
 2372 ** 80  9 LINK= 2384
NUMERO LINEA:  60
 152      52      58      160     52      58
 144      0
 2384 ** 0  0
```

Come puoi vedere abbiamo stampato le linee di programma andando a capo ad ogni linea e indicando l'indirizzo di inizio di ogni istruzione, che coincide con il numero contenuto nei due byte di link. Puoi andare a cercare nelle tabelle dei codici delle parole chiave e dei caratteri e ricostruire le linee del programma.

Il sistema usa dei puntatori per seguire lo svolgimento del programma, e precisamente:

byte 57/58 numero linea corrente
byte 59/60 numero ultima linea del programma precedente
byte 61/62 indirizzo linea corrente

più altri che puoi ritrovare nell'elenco delle variabili nel prossimo paragrafo.

Segue il programma NUMLINEE, che memorizza in una matrice di 5 righe e 3 colonne (non usiamo gli indici 0), lo stato dei tre puntatori citati, e dopo stampa la matrice.

```
1 REM NUMLINEE
5 DEFFNA(B)=PEEK(B)+256*PEEK(B+1)
6 Y=57:Z=59:T=61
7 OPEN4,4:CMD4:PRINT"CONTENUTO PUNTATORI"
8 PRINT
9 PRINT"57/58      59/60      61/62":PRINT
10 DIMX(5,3)
15 X(1,1)=FNA(Y)
16 X(1,2)=FNA(Z)
17 X(1,3)=FNA(T)
18 X(2,1)=FNA(Y)
19 X(2,2)=FNA(Z)
20 X(2,3)=FNA(T)
21 X(3,1)=FNA(Y)
22 X(3,2)=FNA(Z)
23 X(3,3)=FNA(T)
24 X(4,1)=FNA(Y)
25 X(4,2)=FNA(Z)
26 X(4,3)=FNA(T)
27 X(5,1)=FNA(Y)
28 X(5,2)=FNA(Z)
29 X(5,3)=FNA(T)
100 FORK=1TO5:FORJ=1TO3
105 PRINTX(K,J);"         ";:NEXTJ:PRINT
110 NEXTK
199 PRINT#4:CLOSE4:STOP
```

RISULTATI PROGRAMMA PRECEDENTE

```
CONTENUTO PUNTATORI

57/58      59/60      61/62

 15         0         2231
 18         0         2282
 21         0         2333
 24         0         2384
 27         0         2435
```

```
CONTENUTO PUNTATORI

57/58        59/60        61/62

 15           199          2231
 18           199          2282
 21           199          2333
 24           199          2384
 27           199          2435
```

```
CONTENUTO PUNTATORI

57/58        59/60        61/62

 15           10          2231
 18           10          2282
 21           10          2333
 24           10          2384
 27           10          2435
```

Abbiamo riportato 3 risultati per controllare il contenuto dei byte 59 e 60. Nel primo risultato il puntatore è zero; il programma è stato fatto girare subito dopo l'accensione del calcolatore. Nel secondo risultato il puntatore contiene 199, che è il numero di linea eseguito per ultimo facendo girare il programma la prima volta. Nel terzo risultato il puntatore contiene 10; esso è stato ottenuto così: dopo il comando RUN abbiamo premuto subito RUN/STOP e il programma si è arrestato con il messaggio BREAK IN LINE 10; subito dopo abbiamo dato ancora RUN.

Questa prova può anche non essere particolarmente interessante, ma serve a indicare un metodo per capire come si comporta il calcolatore.

Facciamo un breve cenno alla sistemazione in memoria dei programmi in linguaggio macchina. Essi occupano una serie di byte, il cui contenuto. binario è il programma in linguaggio macchina. Se non disponi di un Programma Assemblatore o di un Monitor o di un Caricatore Esadecimale, devi scrivere con POKE a partire dall'indirizzo voluto il contenuto dei byte, usando il corrispondente valore decimale.

8.4 MEMORIA DI LAVORO DEL SISTEMA

Le prime quattro pagine di memoria RAM sono usate dall'Interprete Basic e dal Sistema Operativo per lavorare. Abbiamo già visto l'utilizzo di alcuni puntatori servendoci di brevi programmi esemplificativi. Sarebbe molto interessante condurre uno studio completo sull'argomento, ma questo manuale diventerebbe troppo lungo. Qui riportiamo l'elenco dei byte delle pagine 0, 1, 2 e 3 della RAM con brevi spiegazioni sul loro utilizzo per tua comodità di consultazione. In alcuni casi potrai preparare dei programmi esempio, sulla scia di quelli suggeriti da noi, e arrivare a una comprensione più completa di specifici argomenti. Noi stessi in questo manuale e nei prossimi approfondiremo alcuni degli argomenti ai quali qui si fa solo un breve cenno.

Nella tabella che segue riportiamo su una riga il nome simbolico della variabile usata dal sistema (LABEL), l'indirizzo decimale del o dei byte, l'indirizzo esadecimale del o dei byte (senza H finale) e sulla riga seguente, ed eventualmente sulle successive, brevi spiegazioni. Quando i byte interessati sono più di uno riportiamo l'indirizzo iniziale e quello finale separati da una barra (/). In alcuni casi nelle spiegazioni compare un numero e poi tra parentesi un altro numero; il primo è decimale, il secondo la conversione in esadecimale.

PAGINE 0, 1, 2, 3 DELLA MEMORIA RAM

D6510 0 0000
Registro direzione dati del circuito 6510. Al momento dell'accensione del calcolatore contiene 47. Tale valore dà la configurazione di memoria riportata nella Figura 1.6 del Capitolo 1.

R6510 1 0001
Registro di Input/Output del circuito 6510. Al momento dell'accensione del calcolatore contiene 55.

2 0002
Non usato

ADRAY1 3/4 0003/0004
Vettore di salto per la conversione floating-point/intero.

ADRAY2 5/6 0005/0006
Vettore di salto per la conversione intero/floating-point.

CHARAC 7 0007
Byte di lavoro: carattere da ricercare.

ENDCHR 8 0008
Indicatore ricerca virgolette di chiusura stringhe.

TRMPOS 9 0009
Posizione carattere sullo schermo dopo TAB.

VERCK 10 000A
Indicatore per lettura programmi: 0=LOAD, 1=VERIFY.

COUNT 11 000B
Puntatore buffer input/numero dimensioni variabile con indici.

DIMFLG 12 000C
Indicatore dimensioni variabile con indice non definita da DIM.

VALTYP 13 000D
Tipo di dato: 255 (FF) per stringa, 0 (00) per numerico.

INTFLG 14 000E
Tipo di dato: 128 (80) per intero, 0 (00) per floating-point.

GARBFL 15 000F
Memoria lavoro uso multiplo.

SUBFLG 16 0010
Indicatore uso multiplo.

INPFLG 17 0011
Indicatore per operazione lettura: 0 (00) per INPUT, 64 (40) per GET, 152 (98) per READ.

TANSGN 18 0012
Segno per funzione TAN/confronto risultati.

19 0013
Indicatore richiesta Input.

LINNUM 20/21 0014/0015
Memoria temporanea per numero intero.

TEMPPT 22 0016
Puntatore stack per stringa in elaborazione.

LASTPT 23/24 0017/0018
Indirizzo ultima stringa in elaborazione.

TEMPST 25/33 0019/0021
Stack per stringa in elaborazione.

INDEX 34/37 0022/0025
Area puntatori per programmi di utilità.

RESHO 38/42 0026/002A
Risultato moltiplicazione floating-point.

TXTTAB 43/44 002B/002C
Puntatore inizio programma Basic; di norma contiene 2049.

VARTAB 45/46 002D/002E
Puntatore inizio variabili programma Basic.

ARYTAB 47/48 002F/0030
Puntatore inizio Array programma Basic.

STREND 49/50 0031/0032
Puntatore fine Array + 1 (ind. byte successivo).

FRETOP 51/52 0033/0034
Puntatore all'indirizzo più alto della memoria utilizzata per i corpi delle stringhe.

FRESPC 53/54 0035/0036
Puntatore ultima stringa memorizzata.

MEMSIZ 55/56 0037/0038
Puntatore all'indirizzo più alto disponibile per il programma Basic; al momento dell'accensione contiene 40960.

CURLIN 57/58 0039/003A
Numero linea corrente del programma presente in memoria.

OLDLIN 59/60 003B/003C
Numero linea eseguita per ultima dal programma precedente, o numero della linea dell'ultimo BREAK del programma corrente.

OLDTXT 61/62 003D/003E
Indirizzo linea corrente del programma corrente (ind. byte dove inizia la linea che ha il numero di linea in CURLIN).

DATLIN 63/64 003F/0040
Numero linea del DATA in utilizzo.

DATPTR 65/66 0041/0042
Indirizzo del byte dove inizia una nuova linea DATA o indirizzo dell'elemento disponibile in una linea già in utilizzo.

INPPTR 67/68 0043/0044
Vettore per la routine di INPUT.

VARNAM 69/70 0045/0046
Nome variabile in uso nel programma Basic.

VARPNT 71/72 0047/0048
Puntatore alla variabile in uso nel programma Basic.

FORPNT 73/74 0049/004A
Puntatore alla variabile di controllo del ciclo FOR/NEXT.

75/96 004B/0060
Area di lavoro ad uso multiplo.

FACEXP 97 0061
Esponente (caratteristica) ACC 1 per calcolo numeri floating-point.

FACHO 98/101 0062/0065
Mantissa ACC 1 per calcolo numeri floating-point.

FACSGN 102 0066
Segno mantissa ACC 1.

SGNFLG 103 0067
Indicatore per la valutazione del segno.

BITS 104 0068
Indicatore Overflow per ACC 1.

ARGEXP 105 0069
Esponente ACC 2.

ARGHO 106/109 006A/006D
Mantissa ACC 2.

ARGSGN 110 006E
Segno ACC 2.

ARISGN 111 006F
Segno risultato confronto tra ACC 1 e ACC 2.

FACOV 112 0070
Ultimo byte (byte basso) ACC 1 usato per l'arrotondamento.

FBUFPT 113/114 0071/0072
Puntatore al buffer della cassetta.

CHRGET 115/138 0073/008A
Sottoprogramma che preleva il prossimo byte dal testo del programma Basic. Il byte 0079 (121) è il punto di entrata per prelevare nuovamente lo stesso byte del testo; esso ha il nome simbolico CHRGOT. Il nome simbolico TXTPTR che corrisponde ai due byte 122 e 123 (007A/007B) si riferisce al puntatore al byte corrente della linea del testo.

RNDX 139/143 008B/008F
Seme usato dalla funzione RND, espresso in floating-point.

STATUS 144 0090
Parola di stato usata dalle Routine KERNAL per Input/Output; nel programma Basic viene chiamata ST.

STKEY 145 0091
Indicatore per tasto STOP/tasto RVS.

SVXT 146 0092
Costante di controllo per velocità nastro.

VERCK 147 0093
Indicatore: 0=LOAD, 1=VERIFY.

C3PO 148 0094
Indicatore per Bus Seriale: carattere bufferizzato per Output.

BSOUR 149 0095
Carattere bufferizzato per Bus Seriale.

SYNO 150 0096
Numero per sincronizzazione cassetta.

151 0097
Area di lavoro.

LDTND 152 0098
Numero file aperti/Puntatore alla tabella dei file.

DFLTN 153 0099
Dispositivo di Input principale (default) 0, cioè tastiera.

DFLTO 154 009A
Dispositivo di Output principale (default) 3, cioè video.

RTY 155 009B
Carattere di parità del nastro.

DPSW 156 009C
Indicatore ricezione byte da nastro.

MSGFLG 157 009D
Indicatore modo operativo: 128 (80) per modo diretto, 0 (00) modo differito.

PTR1 158 009E
Indicatore errore passo 1 nastro.

PTR2 159 009F
Indicatore errore passo 2 nastro.

TIME 160/162 00A0/00A2
Orologio aggiornato automaticamente ogni sessantesimo di secondo.

163/164 00A3/00A4
Area di lavoro.

CNTDN 165 00A5
Contatore in decremento per sincronizzazione cassetta.

BUFPNT 166 00A6
Puntatore al buffer del nastro.

INBIT 167 00A7
Input bit da RS232/Cassetta.

BITCI 168 00A8
Contatore bit Input da RS232/Cassetta.

RINONE 169 00A9
Indicatore per RS232: controllo bit di START.

RIDATA 170 00AA
Buffer byte Input per RS232/Cassetta.

RIPRTY 171 00AB
Parità per Input RS232/Contatore corto cassetta.

SAL 172/173 00AC/00AD
Puntatore Buffer Nastro/ Scrolling video.

EAL 174/175 00QE/00AF
Indirizzo fine nastro/fine programma.

CMPO 176/177 00B0/00B1
Costanti per misura operazioni nastro.

TAPE1 178/179 00B2/00B3
Puntatore inizio buffer nastro.

BITTS 180 00B4
Contatore bit Output RS232/Cassetta.

NXTBIT 181 00B5
Prossimo bit da inviare per RS232/Indicatore fine nastro.

RODATA 182 00B6
Buffer Output RS232.

FNLEN 183 00B7
Lunghezza del nome del file corrente.

LA 184 00B8
Numero logico del file corrente.

SA 185 00B9
Indirizzo secondario file corrente.

FA 186 00BA
Numero logico periferica corrente.

FNADR	187/188	00BB/00BC

Puntatore al nome del file corrente.

ROPRTY	189	00BD

Parità Output per RS232/Cassetta.

FSBLK	190	00BE

Contatore blocco Input/Output cassetta.

MYCH	191	00BF

Buffer seriale.

CAS1	192	00C0

Arresto motore nastro.

STAL	193/194	00C1/00C2

Indirizzo partenza Input/Output.

MEMUSS	195/196	00C3/00C4

Memoria temporanea LOAD nastro.

LSTX	197	00C5

Tasto corrente premuto: CHR$(n); 0 per nessun tasto.

NDX	198	00C6

Numero caratteri presenti nel buffer della tastiera.

RVS	199	00C7

Indicatore stampa caratteri inversi: 1 per si, 0 per no.

INDX	200	00C8

Puntatore alla fine della linea logica in INPUT.

LXSP	201/202	00C9/00CA

Posizione X-Y del cursore all'inizio dell'Input.

SFDX	203	00CB

Indicatore tasto SHIFT premuto.

BLNSW	204	00CC

Abilitazione lampeggio cursore: 0 per lampeggio.

BLNCT	205	00CD

Contatore in decremento per lampeggio.

GDBLN 206 00CE
Carattere sotto il cursore.

BLNON 207 00CF
Indicatore ultima impostazione cursore: lampeggio si/no.

CRSW 208 00D0
Indicatore INPUT/GET da tastiera.

PNT 209/210 00D1/00D2
Puntatore linea corrente del video.

PNTR 211 00D3
Posizione cursore nella linea corrente.

QTSW 212 00D4
Indicatore: Editor in modo "tra virgolette"; 0 (00) per no.

LNMX 213 00D5
Lunghezza fisica linea video.

TBLX 214 00D6
Numero linea attuale del cursore.

215 00D7
Area di lavoro.

INSRT 216 00D8
Indicatore modo inserimento: se > 0 dà il numero inserimenti.

LDTB1 217/242 00D9/00F2
Tabella dei collegamenti per le linee video/Memoria di lavoro per Editor.

USER 243/244 00F3/00F4
Puntatore locazione corrente della RAM colore del video.

KEYTAB 245/246 00F5/00F6
Vettore tabella decodifica tastiera.

RIBUF 247/248 00F7/00F8
Puntatore Buffer Input RS232.

ROBUF 249/250 00F9/00FA
Puntatore Buffer Output RS232.

FREKZP 251/254 00FB/00FE
Spazio libero in pagina 0 per programmi utente.

BASKZP 255 00FF
Area lavoro Basic.

256/511 0100/01FF
STACK AREA del sistema (256 byte). Da 256 a 266 (0100/010A) serve per conversione da numero a stringa. Da 256 a 318 (0100/013E) serve per gli errori di Input da nastro; è chiamato simbolicamente BAD.

BUF 512/600 0200/0258
Buffer INPUT del sistema (89 byte).

LAT 601/610 0259/0262
Tabella KERNAL: numeri logici file attivi.

FAT 611/620 0263/026C
Tabella KERNAL: numeri periferiche dei file attivi.

SAT 621/630 026D/0276
Tabella KERNAL: indirizzi secondari file attivi.

KEYD 631/640 0277/0280
Buffer della tastiera, 10 byte gestiti in modo FIFO (First IN First Out - il primo che entra è il primo che esce). La coda è gestita dal puntatore in 198. Se 198 contiene 0, la coda è vuota.

MEMSTR 641/642 0281/0282
Puntatore parte bassa della memoria per Sistema Operativo; inizialmente contiene 2048.

MEMSIZ 643/644 0283/0284
Puntatore parte alta della memoria per Sistema Operativo; inizialmente contiene 40960.

TIMOUT 645 0285
Indicatore Timeout per IEEE.

COLOR 646 0286
Codice colore caratteri correnti.

GDCOL 647 0287
Colore dello sfondo sotto il cursore.

HIBASE 648 0288
Indirizzo pagina della mappa video; inizialmente contiene 4 (4x256=1024).

XMAX 649 0289
Dimensione del buffer della tastiera; inizialmente contiene 10.

RPTFLG 650 028A
Indicatore ripetizione automatica tasti premuti; 128 (80) per ON.

KOUNT 651 028B
Contatore velocità ripetizione.

DELAY 652 028C
Ritardo ripetizione.

SHFLAG 653 028D
Indicatore tasto premuto, distingue tra SHIFT, CBM, CTRL. 0 nessun tasto, 1 SHIFT, 2 CBM, 4 CTRL, 6 CBM+CTRL, 7 per tutti.

LSTSHF 654 028E
Ultima configurazione ottenuta con il tasto SHIFT.

KEYLOG 655/656 028F/0290
Vettore determinazione tabella tastiera.

MODE 657 0291
Indicatore: 0 (00) disabilita tasto SHIFT, 128 (80) abilita tasto SHIFT.

AUTODN 658 0292
Indicatore scrolling automatico verso il basso: 0 corrisponde a ON.

M51CTR 659 0293
RS232: immagine registro di controllo 6551.

M51CDR 660 0294
RS232: immagine registro di comando 6551.

M51AJB 661/662 0295/0296
RS232: BPS USA non standard (tempo/2-100).

RSSTAT 663 0297
RS232: Immagine registro di stato 6551.

BITNUM 664 0298

RS232: numero di bit che rimangono da inviare.

BAUDOF 665/666 0299/029A
RS232: velocità di trasmissione (baud rate); tempo per un bit completo in microsecondi.

RIDBE 667 029B
RS232: puntatore alla fine del buffer di Input.

RIDBS 668 029C
RS232: inizio (pagina) buffer di Input.

RODBS 669 029D
RS232: inizio (pagina) del buffer di Output.

RODBE 670 029E
RS232: puntatore alla fine del buffer di Output.

IRQTMP 671/672 029F/02A0
Contiene il vettore IRQ durante I/O nastro.

ENABL 673 02A1
Abilita RS232.

674 02A2
Controllo sensore cassetta durante I/O nastro.

675 02A3
Memoria temporanea durante lettura nastro.

676 02A4
Indicatore temporaneo IRQ durante lettura nastro.

677 02A5
Indicatore temporaneo indice linea.

678 02A6
Indicatore televisione: 0=NTSC, 1=PAL.

679/767 02A7/02FF
Non usato.

IERROR 768/769 0300/0301
Vettore per stampa messaggi errore Basic.

IMAIN 770/771 0302/0303
Vettore partenza a caldo del Basic.

ICRNCH 772/773 0304/0305
Vettore ingresso routine TOKEN.

IQPLOP 774/775 0306/0307
Vettore ingresso routine LIST.

IGONE 776/777 0308/0309
Vettore ingresso routine ricerca parole chiave.

IEVAL 778/779 030A/030B
Vettore ingresso valutazione Token.

SAREG 780 030C
Memorizzazione registro A 6502.

SXREG 781 030D
Memorizzazione registro X 6502.

SYREG 782 030E
Memorizzazione registro Y 6502.

SPREG 783 030F
Memorizzazione registro SP 6502.

USRPOK 784 0310
Salto funzione USR.

USRADD 785/786 0311/0312
Indirizzo USR LO/HI.

787 0313
Non usato.

CINV 788/789 0314/0315
Vettore interrupt IRQ.

CBINV 790/791 0316/0317
Vettore interrupt BRK.

NMINV 792/793 0318/0319
Vettore interrupt non mascherabile.

IOPEN 794/795 031A/031B
Vettore ingresso routine KERNAL OPEN.

ICLOSE 796/797 031C/031D
Vettore ingresso routine KERNAL CLOSE.

ICHKIN 798/799 031E/031F
Vettore ingresso routine KERNAL CHKIN.

ICKOUT 800/801 0320/0321
Vettore ingresso routine KERNAL CHKOUT.

ICLRCH 802/803 0322/0323
Vettore ingresso routine KERNAL CLRCHN.

IBASIN 804/805 0324/0325
Vettore ingresso routine KERNAL CHRIN.

IBSOUT 806/807 0326/0327
Vettore ingresso routine KERNAL CHROUT.

ISTOP 808/809 0328/0329
Vettore ingresso routine KERNAL STOP.

IGETIN 810/811 032A/032B
Vettore ingresso routine KERNAL GETIN.

ICLALL 812/813 032C/032D
Vettore ingresso routine KERNAL CLALL.

USRCMD 814/815 032E/032F
Vettore ingresso routine definita dall'utente.

ILOAD 816/817 0330/0331
Vettore ingresso routine KERNAL LOAD.

ISAVE 818/819 0332/0333
Vettore ingresso routine KERNAL SAVE.

820/827 0334/033B
Non usato.

TBUFFR 828/1019 033C/03FB
Buffer cassetta, 192 byte.

1020/1023 03FC/03FF

Non usato.

Partendo dalle indicazioni di questa tabella puoi fare diverse prove, ma devi essere armato di pazienza!

Riportiamo un esempio semplice che riguarda il byte 650: esso è di norma a zero e, di conseguenza, i tasti non hanno la funzione di ripetizione automatica, escludendo quei pochi che sono sempre abilitati alla ripetizione, come la barra di spazio. Segue il programma RIPETAUT, che inizialmente stampa il contenuto del byte 650, chiede una stringa in Input e la stampa, poi modifica il contenuto di 650, chiede una stringa in Input e la stampa. Nel secondo hai la ripetizione automatica per qualunque tasto premi e mantieni premuto.

```
1 REM RIPETAUT
5 POKE650,0
7 PRINT"BYTE 650: ";PEEK(650)
10 INPUTA$
20 PRINTA$
30 POKE650,128
35 PRINT"BYTE 650: ";PEEK(650)
40 INPUTA$
50 PRINTA$
55 POKE650,0:STOP
```

Segue il programma PUNT1 con il quale puoi stampare il contenuto di qualunque byte e puoi introdurre una stringa descrittiva per preparare delle tabelline a scopo di studio. Per uscire dal programma devi rispondere solo con RETURN alla richiesta della descrizione.

```
1 REM PUNT1
3 INPUT"RISULTATI SU STAMPANTE? (S/N) ";R$
4 PRINT:IFR$="N"THEN8
5 OPEN4,4
6 PRINT#4,"***CONTENUTO PUNTATORE A 1 BYTE***"
7 PRINT#4
8 D$="":INPUT"DESCRIZIONE PUNTATORE: ";D$
9 IFD$=""THEN40
10 INPUT"IND. BYTE: ";B
20 PRINTD$;SPC(2);"(";B;")";PEEK(B)
```

```
25 IFR$="N"THEN35
30 PRINT#4,D$;SPC(2);"(";B;")";PEEK(B)
35 PRINT#4:GOTO8
40 IFR$="S"THENCLOSE4
45 STOP
```

Segue il programma PUNT2 con il quale puoi stampare il contenuto dei puntatori a due byte accompagnandolo con una descrizione; anche questo è utile per studiare il comportamento del sistema.

```
1 REM PUNT2
3 INPUT"▮▮RISULTATI SU STAMPANTE? (S/N) ";R$
4 PRINT:IFR$="N"THEN8
5 OPEN4,4
6 PRINT#4,"***CONTENUTO PUNTATORI A 2 BYTE***"
7 PRINT#4
8 D$="":INPUT"DESCRIZIONE PUNTATORE: ";D$
9 IFD$=""THEN40
10 INPUT"IND. BYTE BASSO: ";B
15 INPUT"IND. BYTE ALTO: ";A
20 X=256*PEEK(A)+PEEK(B)
21 PRINTD$;SPC(2);"(";B;"/";A;")";X
25 IFR$="N"THEN35
30 PRINT#4,D$;SPC(2);"(";B;"/";A;")";X
31 PRINT#4
35 GOTO8
40 IFR$="S"THENCLOSE4
45 STOP
```

Infine ecco il programma ZONEMEM; esso serve per stampare, byte dopo byte, il contenuto di zone di memoria, premettendo una breve frase. A scopo di esempio l'abbiamo fatto girare per stampare il contenuto dei 256 byte dell'area STACK.

```
1 REM ZONEMEM
2 INPUT"▮▮RISULTATI SU STAMPANTE? (S/N) ";R$
3 PRINT:IFR$="N"THEN5
4 OPEN4,4
5 D$="":INPUT"DESCRIZIONE ZONA: ";D$
7 IFD$=""THEN40
10 INPUT"IND. PRIMO BYTE: ";B
```

```
15 INPUT"IND. SECONDO BYTE: ";A
20 PRINTD$;SPC(2);"(";B;"/";A;")"
21 C=0:FORJ=BTOA:PRINTPEEK(J);"  ";
22 C=C+1:IFC=6THENPRINT:C=0
23 NEXTJ:PRINT
25 IFR$="N"THEN35
26 PRINT#4,"***CONTENUTO ZONE DI MEMORIA***"
27 PRINT#4
30 PRINT#4,D$;SPC(2);"(";B;"/";A;")":PRINT#4
31 C=0:FORJ=BTOA:PRINT#4,PEEK(J);
32 C=C+1:IFC=8THENPRINT#4:C=0
33 NEXTJ:PRINT#4
35 GOTO5
40 IFR$="S"THENCLOSE4
45 STOP
```

```
***CONTENUTO ZONE DI MEMORIA***

STACK SISTEMA  ( 256 / 511 )

 32  51  49  0  48  48  48  48
 48  48  48  255  0  0  255  255
 0  0  255  255  0  0  255  255
 0  0  255  255  0  0  255  255
 0  0  251  255  0  0  255  255
 232  0  255  255  0  0  252  255
 0  0  255  255  0  0  255  255
 0  0  255  255  0  0  255  255
 0  0  255  255  0  0  255  255
 0  0  255  255  0  0  255  255
 0  0  255  255  0  0  255  255
 0  0  255  255  0  0  255  255
 0  0  255  255  0  0  255  255
 0  0  255  255  0  0  255  255
 0  0  255  255  0  0  255  255
 0  0  255  255  0  0  255  255
 0  0  255  255  200  0  255  255
 0  0  255  255  0  0  255  255
 0  0  255  255  0  0  255  255
 0  0  255  255  0  0  255  255
 0  0  251  255  0  0  255  255
```

```
232  0  255  255  0  0  255  255
0  0  255  255  0  0  255  255
0  0  255  255  0  0  255  255
0  0  255  255  0  125  125  125
234  0  1  0  125  14  1  14
188  97  125  189  172  176  183  188
3  184  21  184  5  0  226  175
179  173  0  183  170  130  170  233
167  129  236  9  129  128  0  0
0  1  137  127  128  0  0  31
0  9  127  167  121  166  156  44
```

L'analisi del contenuto dell'area stack, non è molto semplice, bisognerebbe conoscere il funzionamento delle routine del sistema. Per esempio, nell'area stack sono memorizzate le indicazioni necessarie per gestire i cicli FOR; infatti tra le variabili Basic si trova la variabile di controllo del ciclo come una normale variabile floating-point. Il sistema per gestire il ciclo FOR ha bisogno del valore dello STEP, del valore finale per il contatore, dell'indirizzo di memoria dove iniziare ogni ciclo e dell'indirizzo di memoria dove andare alla fine del ciclo.

8.5 CONFIGURAZIONI

Il COMMODORE 64 dispone di 64K di memoria, più altri 20K di memoria, diciamo, ombra. Di fatto il calcolatore lavora indirizzando solo 64K, ma è possibile scambiare tra loro due blocchi da 8K e uno da 4K (8+8+4=20). Facciamo riferimento alla Figura 8.2. In essa la memoria è rappresentata da un rettangolo suddiviso in blocchi; a sinistra sono scritti gli indirizzi decimali e, tra parentesi, esadecimali di inizio e fine di ogni blocco. All'interno di ogni blocco compaiono delle diciture che specificano il possibile uso del blocco. Tre blocchi sono contrassegnati da un asterisco a destra; essi sono i blocchi "doppi". Essi sono effettivamente doppi, cioè esistono ambedue i blocchi, corrispondono agli stessi indirizzi, ma sono attivi alternativamente, o l'uno o l'altro.

Per quanto riguarda il blocco di 8K che va da 32768 a 40959 (8000-9FFF) il discorso è diverso; esso è presente nel calcolatore come RAM, ma, usando la porta di espansione che comunica con l'esterno sul retro del calcolatore, si può inserire una ROM, che va a sostituirsi agli stessi indirizzi della RAM escludendola.

Nella Figura 1.6 del Capitolo 1 si è mostrata la configurazione della memoria al momento dell'accensione del calcolatore; in essa i 4K da 53248 a 57343 (D000-

65535 (FFFF) 57344 (E000)	8K ROM KERNAL O 8K RAM	•
57343 (DFFF) 53248 (D000)	4K RAM INPUT/OUTPUT O 4K ROM CARATTERI	•
53247 (CFFF) 49152 (C000)	4K RAM	
49151 (BFFF) 40960 (A000)	8K ROM BASIC O 8K RAM O 8K ROM INSERITA DALL'ESTERNO	•
40959 (9FFF) 32768 (8000)	8K RAM O 8K ROM INSERITA DALL'ESTERNO	
32767 (7FFF) 16384 (4000)	16 K RAM	
16383 (3FFF) 0 (0000)	16 K RAM	

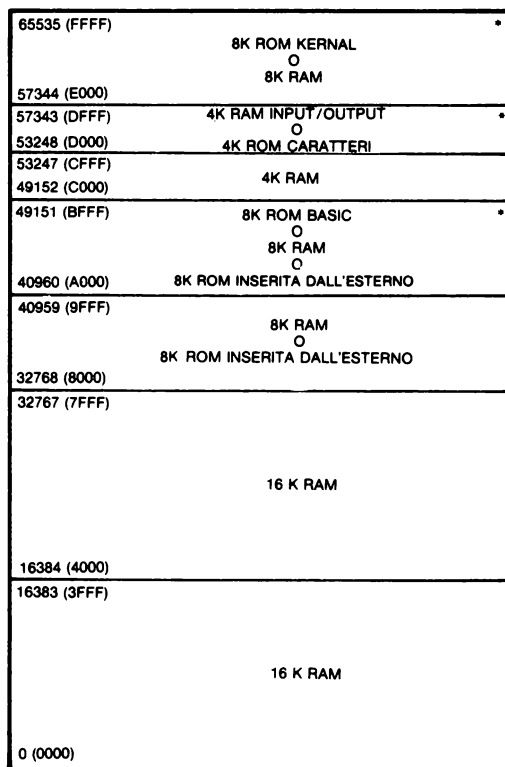



Figura 8.2 Mappa della Memoria

DFFF) sono stati indicati come I/O. Poi nel Capitolo 7 abbiamo visto che da quegli stessi indirizzi si possono andare a prelevare i caratteri in ROM, usando però un particolare accorgimento nel programma (DISECAR). In effetti, con la tecnica della memoria ombra, è possibile associare a questo spazio di 4K indirizzi o una zona di memoria ROM o gli indirizzi dei dispositivi di I/O utilizzati dal sistema.
Vediamo ora più da vicino come si stabilisce la configurazione della memoria.
I primi 2 indirizzi di memoria (Ø e 1) corrispondono a 2 registri interni della CPU che servono appunto per definire la configurazione della memoria attiva.
All'indirizzo Ø si trova il DATA DIRECTION REGISTER (D6510), all'indirizzo 1 si trova il REGISTRO DI I/O (R6510).
A questi 2 registri fanno capo 6 linee della CPU che, opportunamente programmate come vedremo tra breve, attivano o disattivano diverse zone di memoria.
Al momento dell'accensione del calcolatore il D6510 contiene 47 e il R6510 contiene 55. Vediamo il significato dei bit di questi importanti registri.

I bit del registro DATA DIRECTION assegnano alle corrispondenti linee la

funzione di INPUT o OUTPUT; precisamente 1 significa OUTPUT e 0 significa INPUT. La configurazione iniziale 47 corrisponde alla seguente disposizione di 1 e 0:

	bit	7	6	5	4	3	2	1	0
D6510		0	0	1	0	1	1	1	1

I bit di posizione 6 e 7 non sono significativi; gli altri assegnano la funzione di Output, salvo il bit di posizione 4 che assegna la funzione di Input.

Le 6 linee che fanno capo ai registri D6510 e R6510 hanno il seguente significato:

NOME	POS.BIT	DIREZIONE	FUNZIONE
LORAM	0	Output	Controllo zona di memoria 40960/49151 (A000/BFFF)
HIRAM	1	Output	Controllo zona di memoria 57344/65535 (E000/FFFF)
CHAREN	2	Output	Controllo zona di memoria 53248/57343 (D000/DFFF)
	3	Output	Linea scrittura cassetta
	4	Input	Switch interrutori cassetta
	5	Output	Controllo motore cassetta

All'inizio questa porta contiene 55, cioè la seguente configurazione di bit:

	bit	7	6	5	4	3	2	1	0
R6510		0	0	1	1	0	1	1	1

e i bit 0, 1 e 2 al valore 1 danno luogo alla configurazione riportata in Figura 1.6.

Vediamo il significato delle linee della porta 1.

La LORAM è la linea che controlla la presenza o meno degli 8K ROM del BASIC nello spazio degli indirizzi del microprocessore. La ROM è attiva se la linea è programmata alta (HIGH). Quando la linea è programmata bassa (LOW) la ROM sparisce ed al suo posto risultano attivi 8K di RAM.

La HIRAM è la linea che controlla la presenza o meno degli 8K di ROM KERNAL

(Sistema Operativo) nello spazio degli indirizzi del microprocessore. La ROM è attiva quando la linea è programmata alta (HIGH). Quando la linea è programmata bassa (LOW) la ROM sparisce e al suo posto risultano attivi 8K di RAM.

La CHAREN controlla la presenza dei 4K di ROM caratteri quando è al valore 0, mentre quando è al valore 1 sono attivi i 4K di RAM dedicati all' I/O. Lo scambio tra questi due blocchi di 4K di memoria avviene tutte le volte che serve; ne abbiamo visto un esempio nel programma DISECAR del Capitolo 7.

Lo spazio degli indirizzi di 4K dedicato agli I/O è il seguente:

53248/54271	(D000/D3FF)	Controllo Video VIC (1024 byte)
54272/55295	(D400/D7FF)	Sintetizzatore Suoni SID (1024 byte)
55296/56319	(D800/DBFF)	RAM colore (1024 byte)
56320/56575	(DC00/DCFF)	Tastiera CIA1 (256 byte)
56576/56831	(DD00/DDFF)	Bus Seriale, Porta Utente RS-232 (256 byte)
56832/57087	(DE00/DEFF)	Slot # 1 (usato per CP/M (256 byte)
57088/57343	(DF00/DFFF)	Slot # 2 (disco) (256 byte)

Nel determinare la configurazione di memoria attiva giocano un ruolo le linee della porta 1 descritte e, inoltre, i PIN 8 e 9 della porta di espansione; il PIN 8 prende il nome di GAME e il PIN 9 di EXROM. Nella configurazione iniziale con la porta di espansione non utilizzata, GAME e EXROM sono a 1

Riportiamo i possibili utilizzi della memoria che risultano dalla situazione delle linee: LORAN, HIRAM, GAME e EXROM, con un breve commento. (Una linea è "alta" quando è a 1 e "bassa" quando è a 0).

LORAM=1, HIRAM=0, GAME=1, EXROM=qualunque

0/16383	(0000/3FFF)	16K RAM
16384/32767	(4000/7FFF)	16K RAM
32768/49151	(8000/BFFF)	16K RAM
49152/53247	(C000/CFFF)	4K RAM
53248/57343	(D000/DFFF)	4K I/O
57344/65535	(E000/FFFF)	8K RAM

sono disponibili 60K di RAM e 4K di I/O, ma è assente sia il Basic che il Sistema Operativo.

LORAM=1, HIRAM=0, GAME=0, EXROM=0

dà la stessa disposizione di cui sopra, ma è inibito l'accesso alla mappa dei caratteri in ROM.

LORAM=0, HIRAM=1, GAME=1, EXROM=qualunque

0/16383	(0000/3FFF)	16K RAM
16384/32767	(4000/7FFF)	16K RAM
32768/49151	(8000/BFFF)	16K RAM
49152/53247	(C000/CFFF)	4K RAM
53248/57343	(D000/DFFF)	4K I/O
57344/65535	(E000/FFFF)	8K ROM

dà 52K di RAM consecutivi, 4K di I/O e le routine KERNAL; è una confuigurazione adatta ad essere usata con altri linguaggi e altri Sistemi Operativi.

LORAM=0, HIRAM=0, GAME=1/qualunque, EXROM=qualunque/0

dà 64K di RAM, ma manca l'I/O e deve essere immesso tutte le volte che serve.

LORAM=1, HIRAM=1, GAME=1, EXROM=0

è la configurazione standard con una ROM esterna (in generale l'espansione Basic) posizionata da 32768 a 40959 (8000/9FFF).

LORAM=0, HIRAM=1, GAME=0, EXROM=0

corrisponde alla configurazione base dove il BASIC è stato sostituito da una ROM di 8K inserita dall'esterno; serve per programmi speciali che lavorano senza il Basic.

LORAM=1, HIRAM=1, GAME=0, EXROM=0

corrisponde alla configurazione base dove è stata inserita una ROM da 16K che si situa da 32768 a 49151 (8000/BFFF) e si sostituisce al BASIC e ad 8K di RAM. Serve per applicazioni speciali tipo Word Processor o altro.

Quando non si cita CHAREN, è possibile porla a zero e attivare la ROM dei caratteri.

8.6 ASSEGNAZIONI MEMORIA PER I/O

Riportiamo l'assegnazione degli indirizzi di memoria dei 4K da 53248 a 57343 (D000H-DFFFH) per l'I/O, quando questa zona è attiva come RAM. Nel seguito non facciamo seguire H ai numeri in esadecimali che riportiamo tra parentesi.

CONTROLLO VIDEO 52248/53294 (D000/D02E)
(VIC) MOS 6566

53248/53249	(D000/D001)	Pos. X/Y Sprite 0
53250/53251	(D002/D003)	Pos. X/Y Sprite 1
53252/53253	(D004/D005)	Pos. X/Y Sprite 2
53254/53255	(D006/D007)	Pos. X/Y Sprite 3
53256/53257	(D008/D009)	Pos. X/Y Sprite 4
53258/53259	(D00A/D00B)	Pos. X/Y Sprite 5
53260/53261	(D00C/D00D)	Pos. X/Y Sprite 6
53262/53263	(D00E/D00F)	Pos. X/Y Sprite 7
53264	(D010)	Pos. X Sprite 0/7, byte alto (MSB)
53265	(D011)	REGISTRO CONTROLLO

BIT 7: comparatore quadro (bit 8, vedi anche 53266)
BIT 6: modo testo colore esteso, 1 abilita
BIT 5: modo Bit-Map, 1 abilita
BIT 4: riempie lo schermo con il colore del bordo
BIT 3: seleziona 24/25 righe testo, 1 per 25 righe
BIT 2/0: scorrimento rallentato fino alla posizione Y del punto
53266 (D012) Lettura/Scrittura valore RASTER per confronto IRQ
53267 (D013) Latch Pos. X penna ottica
53268 (D014) Latch Pos. Y penna ottica
53269 (D015) Abilita animazione video, 1 abilita
53270 (D016) REGISTRO CONTROLLO
BIT 7/6: non usati
BIT 5: DEVE ESSERE SEMPRE ZERO
BIT 4: modo Multicolor, 1 abilita
BIT 3: seleziona 38/40 colonne video, 1 abilita
BIT 2/0: Scorrimento rallentato fino alla posizione X
53271 (D017) espansione verticale Sprite 0/7, 2 volte Y
53272 (D018) REGISTRO CONTROLLO MEMORIA
BIT 7/4: indirizzo base della matrice video all'interno del VIC
BIT 3/0: indirizzo base di un punto/carattere al'interno del VIC
53273 (D019) REGISTRO INDICATORE INTERRUZIONI, 1 per IRQ verificato
BIT 7: 1 per qualunque IRQ
BIT 3: indicatore IRQ penna ottica
BIT 2: indicatore IRQ collisione tra Sprite

BIT 1: indicatore IR collisione Sprite/sfondo
BIT 0: indicatore IRQ confronto RASTER
53274 (D01A) REGISTRO MASCHERA IRQ, 1 abilita interruzione
53275 (D01B) priorità Sprite/sfondo,1 per Sprite
53276 (D01C) modo Multicolor per Prite 0/7, 1 abilita
53277 (D01D) espansione orizzontale Sprite 0/7, 2 volte X
53278 (D01E) riconoscimento contatto tra due Sprite
53279 (D01F) riconoscimento contatto Sprite/sfondo
53280 (D020) colore bordo
53281 (D021) colore sfondo 0
53282 (D022) colore sfondo 1
53283 (D023) colore sfondo 2
53284 (D024) colore sfondo 3
53285 (D025) registro 0 Sprite Multicolor
53286 (D026) registro 1 Sprite Multicolor
53287/53294 (D027/D02E) colore Sprite da 0 a 7

SINTETIZZATORE SUONI 54272/54300 (D400/D41C)
(SID) MOS 6581

54272 (D400) controllo frequenza voce 1 (byte LO)
54273 (D401) controllo frequenza voce 1 (byte HI)
54274 (D402) ampiezza onda quadra (byte LO)
54275 (D403) ampiezza onda quadra (byte HI)
BIT 7/4: non usati
BIT 3/0: semibyte HI
54276 (D404) REGISTRO CONTROLLO VOCE 1
BIT 7: seleziona forma onda Rumore, 1 abilitato
BIT 6: seleziona onda quadra, 1 abilitato
BIT 5: seleziona onda "dente di sega", 1 abilitato
BIT 4: seleziona onda triangolare, 1 abilitato
BIT 3: di controllo,1 disabilita oscillatore 1
BIT 2: modulatore anello oscillatore 1 con uscita oscillatore 3, 1 abilitato
BIT 1: sincronizzazione frequenza oscillatore1/oscillatore 3, 1 abilitato
BIT 0: controllo porta, 1 attiva Attacco/Decadimento/Sostegno, 0 attiva rilascio
54277 (D405) generatore inviluppo 1 per Attacco/Decadimento ciclo
BIT 7/4: seleziona ciclo Attacco durata 0/15
BIT 3/0: seleziona ciclo Decadimento durata 0/15
54278 (D406) generatore inviluppo 1 per Sostegno/Rilascio
BIT 7/4: seleziona ciclo Sostegno durata 0/15
BIT 3/0: seleziona ciclo Rilascio durata 0/15
54279/54283 (D407/D40B) come 54272/54276, ma per voce 2
54284/54285 (D40C/D40D) come 54277/54278, ma per generatore 2 inviluppo

54286/54290 (D40E/D412) come 54272/54276, ma per voce 3
54291/54292 (D413/D414) come 54277/54278, ma per generatore 3 inviluppo
54293 (D415) frequenza taglio del filtro, semibyte LO (BIT 2/0)
54294 (D416) frequenza taglio del filtro, byte HI
54295 (D417) controllo risonanza filtro/ingresso voce
BIT 7/4: selezione risonanza filtro 0/15
BIT 3: ingresso esterno filtro, 1 per SI, 0 per NO
BIT 2: uscita filtro voce 3, 1 per SI, 0 per NO
BIT 1: uscita filtro voce 2, 1 per SI, 0 per NO
BIT 0: uscita filtro voce 1, 1 per SI, 0 per NO
54296 (D418) selezione modo e volume filtro
BIT 7: taglio uscita voce 3, 1 per OFF, 0 per ON
BIT 6: selezione filtro passa-alto, 1 per ON
BIT 5: selezione filtro passa-banda, 1 per ON
BIT 4: seleziona filtro passa-basso, 1 per ON
BIT 3/0: selezione volume uscita 0/15
54297 (D419) convertitore analogico/digitale Paddle 1, 0/255
54298 (D41A) convertitore analogico/digitale Paddle 2, 0/255
54299 (D41B) generatore numeri a caso oscillatore 3
54300 (D41C) uscita generatore inviluppo 3

MAPPA COLORE 55296/56319 (D800/DBFF)

contiene i colori corrispondenti alle posizioni della MAPPA VIDEO

ADATTATORE INTERFACCIA COMPLESSA CIA 1 56320/56335 (DC00/DC0F)
MOS 6526

56320 (DC00) porta A dati (tastiera, joystick, paddle)
BIT 7/0: scrive valori colonna scansione tastiera
BIT 7/6: legge paddle, 01 per porta A, 10 per porta B
BIT 4: pulsante sparo joystick A, 1 per fuoco
BIT 3/2: pulsanti sparo paddle
BIT 3/0: direzione joystick A, 0/15
56321 (DC01) porta B dati: (tastiera, joystick, paddle)
BIT 7/0: legge valori colonna scansione tastiera
BIT 7: timer B, uscita impulso
BIT 6: timer A, uscita impulso
BIT 4: pulsante sparo joystick 1, 1 per fuoco
BIT 3/2: pulsanti sparo paddle
BIT 3/0: direzione joystick 1
56322 (DC02) REGISTRO DIREZIONE DATI Porta A

56323 (DC03) REGISTRO DIREZIONE DATI Porta B
56324 (DC04) timer A, byte LO
56325 (DC05) timer A, byte HI
56326 (DC06) timer B, byte LO
56327 (DC07) timer B, byte HI
56328 (DC08) orologio, contatore decimi secondo
56329 (DC09) orologio, contatore secondi
56330 (DC0A) orologio, contatore minuti
56331 (DC0B) orologio, contatore ore BIT 6/0, BIT 7 indicatore AM/PM
56332 (DC0C) buffer dati I/O seriale sincrono
56333 (DC0D) REGISTRO CONTROLLO INTERRUZIONI CIA (lettura IRQ/ maschera scrittura)
BIT 7:Indicatore IRQ, 1 segnala interruzione
BIT 4: indicatore 1 IRQ, lettura cassetta/ input SRQ bus seriale
BIT 3: interruzione porta seriale
BIT 2: interruzione orologio
BIT 1: interrruzione timer B
BIT 0: interruzione timer
56334 (DC0E) REGISTRO A CONTROLLO CIA
BIT 7: frequenza orologio, 1 per 50 Hz, 0 per 60 Hz
BIT 6: modo porta seriale, 1 per Output, 0 per Input
BIT 5: contatore timer A, 1 per segnali CNT, 0 per clock 2 sistema
BIT 4: caricamento forzato timer A, 1 per SI
BIT 3: modo funzionamento timer A, 1 per monostabile, 0 per continuo
BIT 2: modo uscita timer A per PB6, 1 per bistabile, 0 per pulsazione
BIT 1: uscita timer A per PB6, 1 per SI, 0 per NO
BIT 0: flag timer a, 1 attiva, 0 ferma
56335 (DC0F) REGISTRO B CONTROLLO CIA
BIT 7: predispone allarme/TOD clock 1, 1 per allarme, 0 per TOD
BIT 6/5: selezione modo timer B:
00 = conteggio pulsazioni clock 02
01 = conteggio transizioni positive CNT
10 = conteggio pulsazioni underflow timer A
11 = conteggio pulsazioni underflow timer A mentre si mantiene positivo CNT
BIT 4/0: come gli stessi bit del registro di controllo A, ma per il timer B

ADATTATORE INTERFACCIA COMPLESSA CIA 2
56576/56591 (DD00/DD0F)
MOS 6526

56576 (DD00) porta dati A (bus seriale, RS-232, controllo memoria VIC)
BIT 7: ingresso bus seriale
BIT 6: ingresso impulsi clock bus seriale

BIT 5: uscita bus seriale
BIT 4: uscita impulsi clock bus seriale
BIT 3: uscita segnale ATN bus seriale
BIT 2: uscita dati RS-232 (porta utente)
BIT 1/0: selezione banco memoria sistema del chip VIC, valore per difetto 11
56577 (DD01) porta dati B (porta utente, RS-232)
BIT 7: Data Set Ready
BIT 6: Clear to Send
BIT 5: utente
BIT 4: Carrier Detect
BIT 3: Ring Indicator
BIT 2: Data Terminal Ready
BIT 1: Request to Send
BIT 0: Received Data
56578 (DD02) REGISTRO DIREZIONE DATI PORTA A
56579 (DD03) REGISTRO DIREZION DATI PORTA B
56580 (DD04) timer A, byte LO
56581 (DD05) timer A, byte HI
56582 (DD06) timer B, byte LO
56583 (DD07) timer B, byte HI
56584 (DD08) orologio, decimi secondo
56585 (DD09) orologio, secondi
56586 (DD0A) orologio, minuti
56587 (DD0B) orologio, ore BIT 6/0, flag AM/PM BIT 7
56588 (DD0C) buffer dati seriali
56589 (DD0D) REGISTRO CONTROLLO INTERRUZIONI CIA (lettura NMI/scrittura maschera)
BIT 7: indicatore NMI, 1 per avvenuto NMI
BIT 4: indicatore 1 NMI, dati input utente/RS-232
BIT 3: interruzione porta seriale
BIT 1: interruzione timer B
BIT 0: interruzione timer A
56590 (DD0E) REGISTRO A CONTROLLO CIA
disposizione BIT come 56334
56591 (DD0F) REGISTRO B CONTROLLO CIA
disposizione BIT come 56335

CAPITOLO 9

ERRORI

9.1 TIPI DI ERRORI

Scrivendo programmi in Basic per il calcolatore si possono commettere diversi tipi di errori.
Gli errori più semplici da correggere sono in generale quelli per i quali il sistema invia un messaggio di segnalazione.
In questa categoria rientrano gli errori di sintassi che commettiamo nello scrivere le frasi del linguaggio. Questo Basic accetta quando si lavora sotto EDITOR le frasi errate, le memorizza, si accorge che sono errate quando tenta di eseguirle. Comunque questi errori sono facili da correggere, al limite andiamo a rivedere sul manuale la sintassi corretta e sistemiamo.
Altri errori che vengono segnalati dal sistema durante l'esecuzione del programma sono quelli che nascono da dati errati e non adatti al tipo di operazione che si vuole eseguire. Tipico errore di questa categoria è quello che si ha quando in un calcolo si imposta una divisione con divisore variabile e questo diventa zero, oppure quando si estrae la radice (funzione SQR) da una espressione che assume valore negativo. Altro errore di questa categoria è arrivare ad usare degli indici che vanno fuori dal range stabilito con la DIM. Oppure scrivere male un file di dati e poi in lettura avere segnalazione di errore. Questi tipi di errori possono essere difficili da correggere se il programma è costruito usando algoritmi complessi.

Tra i messaggi del sistema ci sono anche quelli "non di errore", ma di "situazione", come: BREAK IN LINE, che qui non consideriamo.

Quando abbiamo parlato dei numeri del calcolatore abbiamo fatto notare che si possono avere errori di arrotondamento nei calcoli; devi fare attenzione e usare prudenza nello stabilire confronti tra i risultati dei calcoli.

Un discorso a parte meritano gli errori di logica che commettiamo quando impostiamo un programma per il calcolatore. Si verifica la situazione che il programma "gira", cioè arriva fino in fondo, ma i risultati che ci fornisce sono molto diversi da

quello che ci aspettavamo. Potrebbe anche succedere che il programma è giusto, i ragionamenti di base per la sua impostazione sono giusti, ma noi abbiamo sbagliato a preparare i risultati dei casi prova! A me questo caso non si è mai presentato, ma a qualcuno sarà sicuramente successo.
Comunque questi tipi di errori sono difficili da sistemare; dobbiamo rivedere tutta l'impostazione del programma e ripercorrere passo a passo il lavoro fatto. Sono circostanze nelle quali, se non si è proceduto in modo ordinato documentando la procedura, si è portati a dare in escandescenze verbali. Se siamo saggi, la prossima volta lavoreremo meglio.

9.2 RICERCA ERRORI NEI PROGRAMMI

Per nostra fortuna il Basic facilita la ricerca degli errori nei programmi. Infatti possiamo porre in una prima stesura parecchi STOP nel programma; ogni volta che durante l'esecuzione viene incontrato uno STOP, il programma si arresta ed esce sul video il messaggio BREAK IN LINE. In tale modo sappiamo dove siamo, possiamo consultare il listato del programma e leggere in modo immediato i contenuti delle variabili che ci interessano al momento. Attenzione però a non entrare in EDITOR, in tale caso si perdono i contenuti delle variabili e bisogna ricominciare da capo.
Quando siamo sicuri che una parte di programma va bene possiamo eliminare gli STOP che sono serviti solo in fase di prova. Per riconoscere facilmente le frasi da togliere dopo le prove possiamo inserire dopo le istruzioni una REM con dei caratteri facilmente riconoscibili sul listato.
Oltre agli STOP, possiamo inserire nel programma delle istruzioni che stampino sul video risultati intermedi nei punti più delicati del programma prima della fermata. In tale modo non è necessario intervenire in modo immediato per ricercare risultati.
Ricordati che dopo uno STOP il programma prosegue se si scrive CONT e si preme RETURN.
Ricordati anche che è meglio avere sempre a portata di mano un listato del programma.

La tecnica dei sottoprogrammi aiuta a scrivere programmi senza errori. Infatti un sottoprogramma svolge in generale un compito ben determinato ed è formato da poche linee di programma facilmente provabili. Per provare un sottoprogramma bastano poche linee di programma principale che lo lanciano e stampano i risultati ottenuti.
E' consigliabile scrivere per prima cosa i sottoprogrammi e provarli usando poche linee di lancio che poi si eliminano facilmente dal programma definitivo.
Risulta molto più difficile provare un programma lungo in un colpo solo; se ci sono errori, può essere difficile localizzarli.

9.3 ERRORI SEGNALATI DAL SISTEMA

Passiamo ad elencare in ordine alfabetico i messaggi del sistema.

.. BAD DATA
Il programma attende da un file aperto su una priferica dati numerici, arrivano invece dati non numerici.

.. BAD SUBSCRIPT
Viene richiamata una variabile con indice che non corrisponde al dimensionamento della stessa.

.. CAN'T CONTINUE
In seguito al comando CONT il programma non può proseguire per una delle seguenti ragioni:
.. non esiste un programma in memoria,
.. si è entrati in Editor e quindi si è perso il riferimento per continuare,
.. il programma non ha ricevuto il comando RUN,
.. è stato segnalato un errore.

.. DEVICE NON PRESENT
La periferica richiesta per una operazione del tipo: OPEN, CLOSE, CMD, INPUT #, GET #, PRINT #, non è presente (può essere solo spenta).

.. DIVISION BY ZERO
Si è tentato di dividere per zero e questo non è consentito.

.. EXTRA IGNORED
Sono stati scritti più dati di quanti richiesti in risposta a un comando di INPUT. Sono accettati tanti dati quante sono le variabili presenti nella lista, gli altri vanno persi, ma il programma continua.

.. FILE NOT FOUND
Non si trova il file sulla periferica: per la cassetta si è arrivati alla segnalazione di fine nastro senza trovare il file richiesto, per il disco il nome del file non compare nell'indice del disco.

.. FILE NOT OPEN
Si cerca di lavorare con istruzioni del tipo: CLOSE, CMD, INPUT #, GET #, PRINT #, su un file che non è stato aperto.

.. FILE OPEN
Si tenta di aprire un file già aperto.

.. FORMULA TOO COMPLEX
Si è usata una espressione di calcolo troppo complicata e l'interprete Basic non riesce a calcolarla. Conviene spezzare l'espressione in due parti e riprovare, o calcolare prima separatamente qualche parentesi e sostituire il nome di una variabile che contiene un risultato parziale.

.. ILLEGAL DIRECT
Si è tentato di usare in modo immediato un comando che non può essere usato in tale modo, come GET, INPUT, DEFFN.

.. ILLEGAL QUANTITY
Si è tentato di usare una funzione con un argomento fuori dai limiti consentiti o che non rispetta le regole inerenti alla funzione. Nella spiegazione delle frasi Basic abbiamo citato spesso questo tipo di errore.

.. LOAD
Durante il caricamento di un programma da nastro si trovano errori sulla prima e/o sulla seconda registrazione.

.. NEXT WITHOUT FOR
Si incontra un NEXT senza aver prima incontrato FOR, oppure la variabile del NEXT non corrisponde a quella dell'ultimo FOR aperto.

.. NOT INPUT FILE
Si tenta di leggere da un file che è stato aperto per scrivere.

.. NOT OUTPUT FILE
Si tenta di scrivere su un file che è stato aperto per leggere oppure che può essere solo di lettura, come la tastiera. Il file Video non dà questo tipo di errore.

.. OUT OF DATA
Si tenta di leggere con READ più dati di quelli disponibili. Si ha questo errore se si preme RETURN sopra il READY. del video, infatti viene interpretato come READ Y.

.. OUT OF MEMORY
Manca memoria per lavorare; le cause possono essere:
.. .si sta scrivendo un programma troppo lungo;
.. .in fase esecutiva le stringhe sono diventate molto lunghe ed è finita la RAM disponibile per i corpi delle stringhe;
.. .non si fa un uso corretto dei cicli FOR/NEXT (non si chiudono bene) o se ne concatenano più di 9;
.. .non si fa un uso corretto delle istruzioni GOSUB/RETURN, per esempio si eseguono dei GOSUB, ma poi mancano i RETURN.

Nei primi due casi manca la RAM disponibile per il Basic, negli ultimi due manca spazio nell'area STACK di lavoro del sistema. Per stabilire di cosa si tratta basta scrivere in immediato:

PRINT FRE(0) e poi RETURN

se il numero che esce non è zero, significa che si è in una delle due ultime circostanze e ci sono errori nella costruzione del programma.

.. OVERLOW
Si sono eseguiti calcoli il cui risultato supera le capacità del calcolatore.

.. REDIM'D ARRAY
Si tenta di usare la istruzione DIM per un ARRAY che è già stato dimensionato.

.. REDO FROM START
Si è risposto a una richiesta di dati numerici con caratteri non validi. Dopo il messaggio ricompare il punto interrogativo di richiesta e il programma attende i dati di nuovo prima di proseguire.

.. RETURN WITHOUT GOSUB
Si incontra un RETURN senza aver prima eseguito un GOSUB.

.. STRING TOO LONG
Si è cercato di sommare delle stringhe e si superano in totale 255 caratteri; oppure si legge da file una stringa con più caratteri di quelli consentiti per INPUT.

.. SYNTAX
Una istruzione non è riconoscibile, può contenere i più diversi errori di scrittura.

.. TYPE MISMATCH
Si usa un tipo di dato errato, stringa al posto di numero o viceversa.

.. UNDEF'D FUNCTION
Si richiama una funzione che non è stata definita.

.. UNDEF'D STATEMENT
Una istruzione cita un numero di linea inesistente nel programma.

.. VERIFY
L'operazione di verifica per un programma su nastro o su disco non dà buon risultato.

I messaggi REDO FROM START e EXTRA IGNORED sono gli unici che non interrompono il programma; essi hanno solo una funzione di segnalazione.

APPENDICE A

LA TASTIERA

Il buffer della tastiera può contenere 10 caratter esso si trova dal byte 631 al byte 640. Il contatore dei caratteri del buffer si trova nel byte 198, il codice dell'ultimo carattere premuto si trova nel byte 197. Porre dei caratteri ordinatamente nel buffer della tastiera, inserire in 198 il numero dei caratteri posti, corrisponde a dare la sequenza di caratteri manualmente sulla tastiera; puoi rivedere la routine RDEL al Paragrafo 4.6.
Puoi scrivere alcune linee di comandi in immediato sul video, partendo dalla posizione HOME, per esempio 5 linee, poi scrivi in immediato con POKE a partire dal byte 631 i codici: 19, 13, 13, 13, 13 ,13 e poi in 198 il numero 6. Le cinque linee vengono eseguite, infatti la tastiera (il suo buffer) viene scandita continuamente e vengono recepiti i caratteri che hai memorizzato; il primo manda a HOME e gli altri sono il RETURN per le linee che si trovano sul video.
La tastiera ha una immagine in memoria sotto forma di una matrice di 8 righe e 8 colonne, nella quale ogni posizione corrisponde a uno dei 64 tasti. Il registro 56320 contiene l'immagine delle linee della matrice tastiera e il registro 56321 quella delle colonne; analizzando il contenuto dei due registri il sistema stabilisce il tasto premuto e ne pone il codice in 197. Quando si preme insieme a un tasto uno dei tasti di controllo RVS, CTRL, COMMODORE o SHIFT il codice viene modificato.

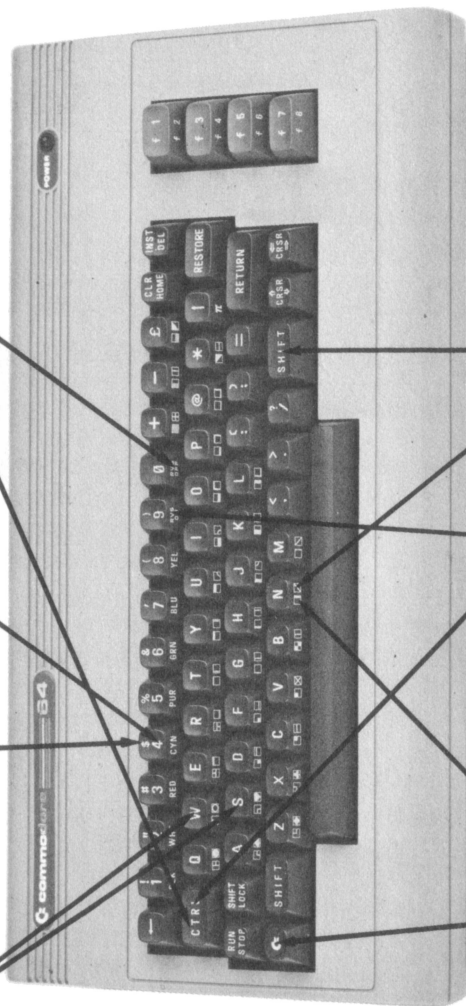
Riteniamo utile presentare alcune immagini della tastiera evidenziando la funzione dei tasti.

Nel set maiuscolo/grafico i tasti normali danno le lettere minuscole. Per l'elenco completo dei caratteri rimadiamo al Capitolo 7.

TASTI NORMALI
SET MAIUSCOLO GRAFICO

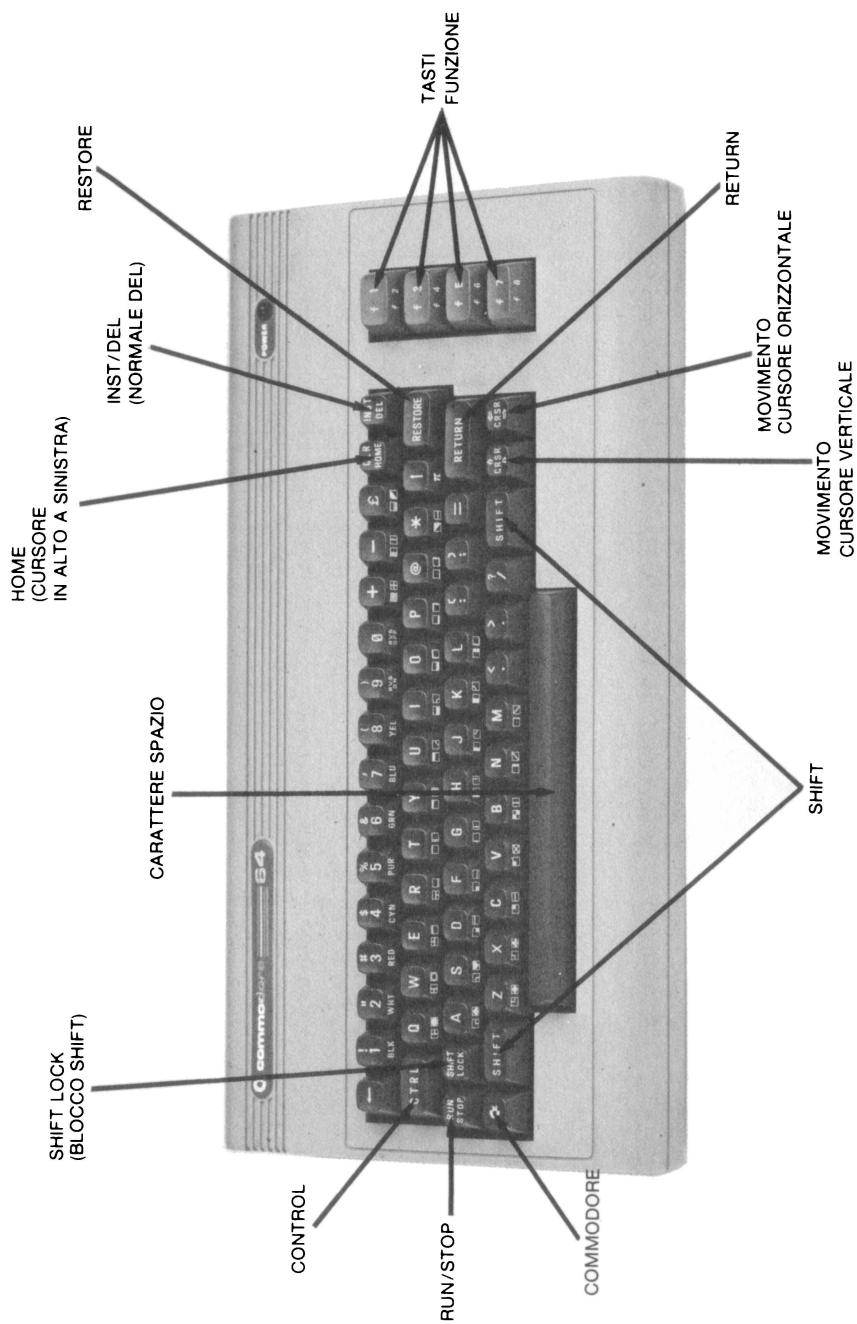
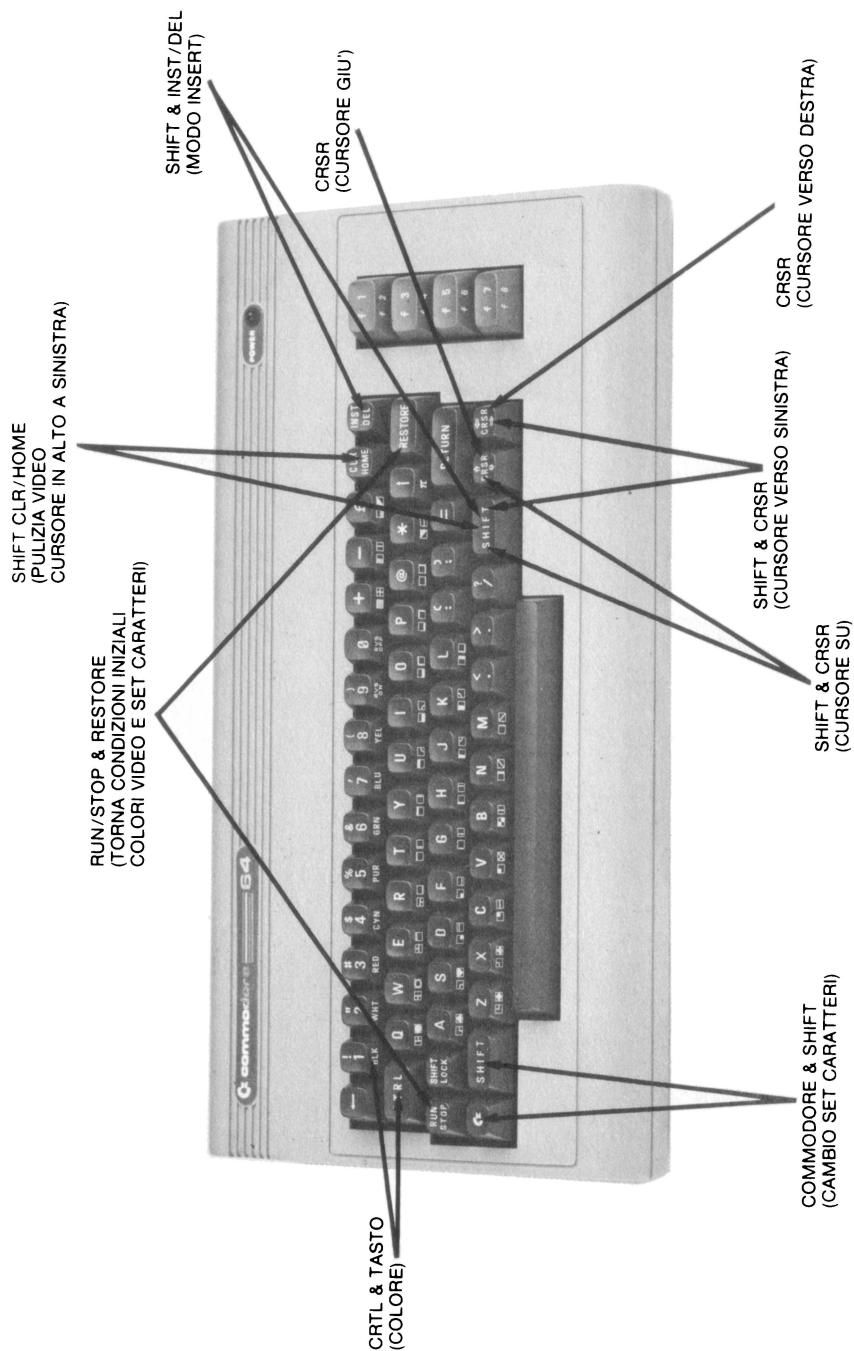
SHIFT & TASTO

TASTO NORMALE

RVS OFF
(CAMPO NORMALE)

RVS ON
(CAMPO INVERSO)

COMMODORE & TASTO

SHIFT & TASTO

APPENDICE B

IL BASIC COMPILATO

E' disponibile un programma di nome PETSPEED 64 - Basic Compiler, distribuito dalla COMMODORE, che consente di compilare programmi scritti in Basic. Il programma risiede su floppy disk ed è fornito di una ROM di protezione, senza la quale non funziona, da inserire nella Porta 2.

Compilare un programma significa avere come risultato un codice intermedio che non gira nel solito modo interpretativo, ma in un altro che è più veloce. Il programma compilato non lavora più in base al nome delle variabili o ai numeri delle linee Basic, ma in base agli indirizzi di memoria. Esso mantiene il nome del programma sorgente, con l'aggiunta di .WOW. Per caricare il programma compilato in memoria non è necessario avere inserita la chiave ROM di attivazione di PEETSPEED; basta scrivere:

LOAD"nome.WOW",8

e il programma viene caricato da disco. Per far girare il programma si scrive RUN, come al solito. Il nome del programma, dopo la compilazione, può essere modificato con la funzione RENAME del DOS.
Il programma compilato non è listabile; se si scive LIST si vede una sola linea di programma. Inoltre esso non può essere interrotto per passare ad eseguire qualche comando in modo immediato; i nomi delle variabili non sono più direttamente accessibili. Il dischetto dei programmi di utilità che si genera nella procedura di compilazione, consente di ottenere utili prestazioni.
Il vantaggio del programma compilato sta nella sua maggior velocità esecutiva. Sul dischetto il programma compilato occupa molti più blocchi di quello originale, infatti esso comprende tutto, istruzioni e variabili e un suo particolare programma interprete.

Noi abbiamo usato il programma su un calcolatore collegato ad una sola unità 1541.

Nel manuale vengono spiegate alcune limitazioni che devono essere rispettate nei programmi da compilare; le riassumiamo brevemente:

.. le istruzioni DIM non accettano variabili per gli indici; si devono usare numeri;
.. i programmi non devono essere troppo lunghi; sembra che 70 blocchi o poco più siano già un limite;
.. il programma deve essere tutto in Basic e quindi non comprendere parti in linguaggio macchina;
.. non si possono fare riferimenti alle locazioni della pagina zero nel solito modo, ma il manuale fornisce le necessarie spiegazioni per modificare il programma;
.. non si può usare RUN numero-linea;
.. non si può usare LIST e SAVE nel programma;
.. un programma non può dividere le variabili con un altro programma da esso richiamato in memoria.

Il compilatore consente operazioni non accettate normalmente in Basic:

.. nei cicli FOR sono accettate variabili di controllo intere;
.. si possono definire funzioni utente che operano su stringhe.

Noi abbiamo preso il programma ORDIN del Capitolo 6; abbiamo apportato le poche modifiche necessarie, e cioè la linea 25 della DIM è stata cambiata dimensionando a 1000, invece che a N-1. Naturalmente così non si possono ordinare più di 1000 numeri e quindi avremmo dovuto aggiungere un controllo dopo la richiesta di N; non l'abbiamo fatto, dato che ci interessava solo un controllo dei tempi di esecuzione. Abbiamo fatto girare il programma originale e il programma compilato, dando lo stesso argomento negativo per l'origine dell'estrazione dei numeri a caso, e lo stesso numero di elementi da ordinare. I risultati sono i seguenti:

.. tra il compilato e il non compilato differiscono i numeri estratti, anche partendo dallo stesso argomento negativo; anche per il compilato la sequenza prodotta è sempre la stessa;
.. il compilato sembra più veloce; il tempo di estrazione è minore;
.. non possiamo fare il paragone dei tempi di ordinamento dato che i due programmi operano su sequenze diverse di numeri.

Riportiamo i risultati della prova dei due programmi, senza la stampa dei numeri.

RISULTATI ORDINA NON COMPILATO

```
BASE RND: -1984
ORDINAMENTO DI  57  NUMERI
```

```
TABELLA NUMERI ESTRATTI

TEMPO ESTRAZIONE 1.7333333  SEC.

TABELLA NUMERI ORDINATI CON METODO  1

TEMPO ORDINAMENTO 35.75  SEC.

TABELLA NUMERI ORDINATI CON METODO  2

TEMPO ORDINAMENTO 19.55  SEC.

TABELLA NUMERI ORDINATI CON METODO  3

TEMPO ORDINAMENTO 13.766667  SEC.
```

RISULTATI ORDINA COMPILATO

```
BASE RND: -1984
ORDINAMENTO DI  57  NUMERI

TABELLA NUMERI ESTRATTI

TEMPO ESTRAZIONE .5  SEC.

TABELLA NUMERI ORDINATI CON METODO  1

TEMPO ORDINAMENTO 3.4  SEC.

TABELLA NUMERI ORDINATI CON METODO  2

TEMPO ORDINAMENTO 2.0833333  SEC.

TABELLA NUMERI ORDINATI CON METODO  3

TEMPO ORDINAMENTO 2.5333333  SEC.
```

Per curiosità riportiamo la tabella dei numeri estratti con argomento − 1984 e N = 57.

COMPILATO

```
TABELLA NUMERI ESTRATTI

 2538      7247      8859      9275      3603
 4016      7106      4361      1522      9440
 4309      6122      2491      7671      8966
 3684      3307      3512      1872      840
 2723      4425      4955      6565      9885
 5385      5418      1659      3636      2614
 4740      1285      125       8932      3063
 6456      7677      7093      7506      3274
 3852      2143      9194      2402      2580
 9945      1575      7864      2624      1813
 3701      1476      4510      354       7930
 7336      3036
TEMPO ESTRAZIONE .5  SEC.
```

NON COMPILATO

```
TABELLA NUMERI ESTRATTI

 1         265       747       883       664
 4072      2955      2889      9985      5283
 8191      4337      384       9694      3658
 5822      9693      2177      9935      4719
 1692      7346      346       8113      3688
 2220      7368      428       6759      7049
 3401      462       9638      3096      6887
 5263      1591      4968      778       6525
 5566      8869      5433      6397      7951
 1120      6169      4422      3728      4065
 7643      5136      2941      558       269
 3482      5984
TEMPO ESTRAZIONE 1.7333333  SEC.
```

Dopo aver esaminato i precedenti risultati, abbiamo deciso di provare a ordinare 300 numeri, ma gli stessi, con un programma Basic e con lo stesso programma compilato. Abbiamo modificato il programma ORDINA ed esso è diventato il programma ORDFISSO; in questo programma, invece di generare una serie di numeri a caso, viene letto un file sequenziale residente su disco, di nome DATI, che contiene 300 numeri. Il file DATI è stato preparato con un semplice programma e memorizzato su disco. In questo modo sia il programma compilato, che quello non compilato, lavorano sugli stessi numeri.
Segue la lista del programma ORDFISSO e i due risultati, solo per quanto riguarda i tempi di esecuzione.

```
1 REM ORDFISSO
2 REM ORDINAMENTO 300 NUMERI
5 REM LETTI DA UN FILE SEQUENZIALE
7 REM DI NOME: DATI
9 S$="         ":X$="ORDINATI CON METODO "
11 Y$="ORDINAMENTO"
14 N=300
17 OPEN4,4:CMD4
19 PRINT"ORDINAMENTO DI ";N;" NUMERI":PRINT
20 PRINT#4:CLOSE4
21 REM NUMERI MINORI UGUALI A 9999
23 INPUT"STAMPA SOLO TEMPI (S/N): ";R$
25 DIM N%(300),P%(300)
30 A=TI
31 OPEN10,8,10,"DATI,S,R"
40 FORK=0TON-1
45 INPUT#10,N%(K):P%(K)=N%(K)
50 NEXTK:B=TI:CLOSE10
55 A$="LETTI":B$="LETTURA DA DISCO":GOSUB100
60 C$=" 1":A$=X$+C$:B$=Y$
63 GOSUB200:GOSUB100
65 C$=" 2":A$=X$+C$:GOSUB500
67 GOSUB400:GOSUB100
69 C$=" 3":A$=X$+C$:GOSUB500
71 GOSUB300:GOSUB100
80 PRINT"BYTE LIBERI: ";FRE(0)
99 STOP
100 OPEN4,4:CMD4
105 PRINT"TABELLA NUMERI "A$:PRINT
107 IFR$="S"THEN125
110 IR=0:FORJR=0TON-1
```

```
115 N$=STR$(N%(JR)):N$=LEFT$(N$+S$,8)
117 PRINTN$;:IR=IR+1
120 IFIR=5THENPRINT:IR=0
123 NEXTJR:PRINT
125 PRINT"TEMPO "B$;(B-A)/60;" SEC."
130 PRINT#4:CLOSE4:RETURN
200 REM METODO A BOLLE
205 L=N-2:A=TI
210 J=0:FORK=0TOL
215 IFN%(K)<=N%(K+1)THEN225
220 C%=N%(K):N%(K)=N%(K+1):N%(K+1)=C%:J=1
225 NEXTK
230 IFJ=0THENB=TI:RETURN
235 L=L-1:GOTO210
300 REM METODO DIMEZZAMENTO INTERVALLO
305 IN=INT(N/2+0.5):A=TI
310 J=0:FORK=0TO(N-1-IN)
315 IFN%(K)<=N%(K+IN)THEN320
317 C%=N%(K):N%(K)=N%(K+IN):N%(K+IN)=C%:J=1
320 NEXTK
325 IFIN=1ANDJ=0THENB=TI:RETURN
327 IFIN=1THEN310
330 IN=INT(IN/2+0.5):GOTO310
400 REM METODO CICLI FISSI
405 K=0:A=TI
410 J=K+1:I=K
415 FORL=JTON-1
420 IFN%(K)<=N%(L)THEN430
425 K=L
430 NEXTL
435 IFK=ITHEN445
440 C%=N%(I):N%(I)=N%(K):N%(K)=C%
445 K=I+1:IFK=N-1THENB=TI:RETURN
450 GOTO410
500 FORK=0TON-1:N%(K)=P%(K):NEXTK:RETURN
```

RISULTATI ORDFISSO NON COMPILATO

```
ORDINAMENTO DI  300  NUMERI


TABELLA NUMERI LETTI

TEMPO LETTURA DA DISCO 9.5  SEC.

TABELLA NUMERI ORDINATI CON METODO  1

TEMPO ORDINAMENTO 979.233333  SEC.

TABELLA NUMERI ORDINATI CON METODO  2

TEMPO ORDINAMENTO 472.533334  SEC.

TABELLA NUMERI ORDINATI CON METODO  3

TEMPO ORDINAMENTO 472.433333  SEC.
```

RISULTATI ORDFISSO COMPILATO

```
ORDINAMENTO DI  300  NUMERI

TABELLA NUMERI LETTI

TEMPO LETTURA DA DISCO 4.7  SEC.

TABELLA NUMERI ORDINATI CON METODO  1

TEMPO ORDINAMENTO 92.95  SEC.

TABELLA NUMERI ORDINATI CON METODO  2

TEMPO ORDINAMENTO 53.3833334  SEC.
```

```
TABELLA NUMERI ORDINATI CON METODO  3

TEMPO ORDINAMENTO 50.2  SEC.
```

Dal confronto si vede che con la versione compilata:

.. il tempo di lettura da disco è dimezzato;
.. il tempo di ordinamento con il metodo 1 è circa 10 volte inferiore;
.. il tempo di ordinamento con il metodo 2 è circa 9 volte inferiore;
.. il tempo di ordinamento con il metodo 3 è circa 9 volte inferiore.

Non riportiamo il listato di ORDINA, dato che abbiamo indicato la modifica apportata rispetto a ORDIN.

Un'accurata esposizione del linguaggio BASIC, accompagnata da numerosi esempi. Un BASIC visto dall'interno. Un libro di programmi per imparare a programmare.

Nel Capitolo 1 si ha una panoramica dei diversi argomenti. Il Capitolo 2 è dedicato al linguaggio; il suo indice riporta in ordine alfabetico le parole chiave del Basic, consentendo una rapida ricerca. Nel Capitolo 3 si approfondisce l'uso della tastiera e del video. Il Capitolo 4 fornisce le informazioni necessarie per usare disco e cassetta per memorizzare programmi, inoltre insegna l'overlay dei programmi e fornisce utili suggerimenti per migliorare il proprio lavoro sul calcolatore. Il Capitolo 5 è dedicato alla stampante e insegna a eseguire la copia del video in diversi modi. Nel Capitolo 6 si parla della costruzione del programma. Nel Capitolo 7 vengono passati in rassegna i codici e i numeri del calcolatore. Il Capitolo 8 è dedicato alla memoria, vengono indicate le possibili configurazioni del COMMODORE 64 e sono riportate le tabelle di assegnazione degli indirizzi. Nel Capitolo 9 si tratta degli errori. Completano il libro, l'Appendice A dedicata alla tastiera e l'Appendice B all'argomento del BASIC compilato.

Tutti i programmi esempio riportati nel libro sono disponibili a richiesta su floppy disk.

L. 26.000 Cod. 348D ISBN 88-7056-036-8

GRUPPO
EDITORIALE
JACKSON

Rita Bonelli

COMMODORE 64 il basic

104